UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INDUSTRIAL: ELECTRÓNICA INDUSTRIAL

PROYECTO FINAL DE CARRERA

SISTEMA DE MEDIDA DE CONSUMO PARA APLICACIONES CRIPTOGRÁFICAS EN MICROCONTROLADORES ARM CORTEX-M3

Autor: **Brandán Currás Paz**

Tutor: **Luis Mengíbar Pozo**

Madrid, Diciembre 2013

Título: Sistema de medida de consumo para aplicaciones criptográficas en microcontroladores ARM Cortex-M3

Autor: Brandán Currás Paz

Director: Luis Mengibar Pozo

La defensa del presente Proyecto Fin de Carrera se realizó el día 20 de Diciembre de 2013, siendo calificada por el siguiente tribunal:

Presidente:

Vocal:

Secretario:

Habiendo obtenido la calificación:

CALIFICACIÓN:

Presidente Secretario Vocal

AGRADECIMIENTOS:

A toda mi familia y a mis seres queridos.

A mi tutor Luis por orientarme en el proyecto y sobre todo por haber sido tan comprensivo a lo largo de este proceso.

ÍNDICE:

1. Introducción y Objetivos

1.1 Introducción

En la actualidad, en la que podríamos definir como la era de las comunicaciones, donde diariamente se realizan millones de intercambios de información en entornos electrónicos, cobra especial importancia el rol de la criptografía por la necesidad de enviar dicha información y que solo sea interpretable por el receptor. Utilidades como el correo electrónico, las redes sociales, compras por internet, y en definitiva casi cualquier actividad que se realiza en Internet, se han convertido en elementos del día a día que intercambian una gran cantidad de información confidencial y es gracias al cifrado de esta información que se puede mantener dicha confidencialidad.

Desde un punto de vista clásico, como se expondrá con más detalle en esta memoria, se consideraba suficiente para proteger un mensaje, un algoritmo criptográfico difícil de *romper*. En las últimas décadas, debido principalmente al desarrollo tecnológico en componentes de medidas y capacidad de procesamiento, se plantea que un mensaje no solo se protege con un buen algoritmo criptográfico sino que también es necesario tener en cuenta la implementación física del *criptosistema* ya que analizando diferentes aspectos físicos es posible romper la codificación. Estos ataques al sistema criptográfico son conocidos como *Ataques de Canal Lateral*

Este proyecto se enmarca en este contexto de los ataques laterales. Se plantea la creación de un entorno de trabajo donde se permita tomar medidas de tiempo y consumo de un sistema criptográfico basado en un microcontrolador ARM.

1.2 Objetivos

Para la realización de este proyecto se ha establecido como objetivo principal la creación un entorno de trabajo que permita realizar medidas de consumo de un dispositivo en la cual se ejecuta una aplicación criptográfica. Este dispositivo es la placa integrada STM32LDiscovery que cuenta con un microcontrolador ARM. Se deberá analizar la capacidad de los componentes integrados de la placa para tomar

medidas de consumo y la viabilidad de este sistema para el estudio en el campo de los ataques de canal lateral.

Para lograrlo es necesaria la implantación del algoritmo criptográfico en el microcontrolador, así como configurar y gestionar los diferentes componentes de la placa que nos permitan realizar las medidas de consumo. Estas medidas deberán ser comunicadas a otro dispositivo que pueda procesar la información. Se usará un ordenador personal para procesar dicha información y también para actuar como interfaz con el usuario.

Es necesario definir una serie de objetivos secundarios dada la necesidad de familiarizarnos con las diferentes herramientas y así poder sacar el máximo rendimiento a la aplicación:

- Estudio del algoritmo AES: Se debe conocer bien el funcionamiento del algoritmo para poder implementarlo en el micro y también para poder realizar la mejora de rendimiento de 32 Bits.
- Estudio de los microcontroladores ARM y el concreto de la familia STM32L. Este estudio es necesario ya que difiere enormemente de los microcontroladores estudiados previamente en la carrera, familia 8051 programados en ensamblador. Esto hace que sea necesario familiarizarse con su arquitectura y forma de programación (lenguaje C).

2.Estado de la técnica

2.1 Criptografía

En esta sección se explica qué es y en qué consiste la criptografía y se realiza un breve repaso a su historia para poner en contexto la importancia que ha tenido, y tiene, desde sus orígenes hasta la actualidad.

2.1.1 Introducción

Uno de los problemas más importantes a lo largo de la Historia en cualquier civilización, ha sido el intercambio de mensajes cifrados, debido a la necesidad de que algunos de éstos solo sean conocidos por aquellas personas a las que van dirigidos y puedan ser interpretados únicamente por ellas. Tanto en el terreno de la economía, diplomacia, espionaje o el militar, constituyen la mejor defensa de las comunicaciones y datos que se intercambian

Hoy en día, con el gran avance en las comunicaciones debido fundamentalmente a la aparición de la informática, la protección de la información es uno de los aspectos más importantes a desarrollar. Teniendo en cuenta que la información es el bien más preciado en estos días, cobra una extraordinaria importancia su ocultación al personal no autorizado.

La utilización de técnicas criptográficas será la única solución para la protección de los datos que intercambian dos o más interlocutores autenticados y cuyo contenido en muchos casos debe mantenerse en secreto por razones personales, empresariales, políticas, militares o de cualquier otra índole. Esto nos permitirá asegurar al menos dos elementos básicos del intercambio de información: la confidencialidad o secreto de la información y la integridad del mensaje, además de la autenticidad del emisor.

2.1.2 Criptografía

La Criptología es la ciencia del estudio de lo oculto. Esta palabra proviene de las griegas *KRYPTO* y *LOGOS*. Una rama de la Criptología es la CRIPTOGRAFÍA, que proviene de *KRYPTOS* y *GRAPHOS* que significan descripción y se ocupa de las técnicas que alteran las representaciones lingüísticas de mensajes, mediante técnicas de cifrado para que no puedan ser entendidos por lectores no autorizados que intercepten esos mensajes [1]. Por tanto, el objetivo de la criptografía es conseguir la confidencialidad de los mensajes. Para ello se diseñan sistemas de cifrado y códigos

Con el desarrollo de la Informática y por tanto de las comunicaciones digitales, han aparecido como un factor importantísimo los problemas de seguridad en las transmisiones que pueden ser fácilmente interceptadas. Hay que garantizar esta seguridad, y este es el objetivo de la criptografía, ciencia que se encarga del estudio de los algoritmos, protocolos y sistemas encaminados a proteger la información, dando seguridad a las comunicaciones y a aquellos que se comunican. Para lograr esta seguridad, los expertos en Criptología investigan y desarrollan, en el terreno de las matemáticas y de la informática, herramientas para lograr este fin.

Naturalmente, frente a la ciencia del cifrado de mensajes (Criptografía), emergió el arte y la ciencia de transgredir las claves de acceso, es decir, la que se encarga de descifrar los mensajes de forma no autorizada: el *criptoanálisis*

2.1.3 Desarrollo de la Criptografía

A pesar del extraordinario desarrollo alcanzado por la Criptografía, debido, fundamentalmente, a la electrónica digital y a la informática, no es un arte o una ciencia exclusivamente moderna. Ha existido desde tiempo inmemorial. Se puede considerar como Criptografía Clásica aquella que se implementó hasta la Segunda Guerra Mundial y Criptografía Moderna a la posterior. En los apartados siguientes se aborda con más detalle las diferentes etapas de la criptografía

2.1.3.1 Criptografía Clásica

Desde los tiempos más remotos y debido a cuestiones políticas, religiosas, militares e incluso económicas, se han utilizado métodos para ocultar o hacer ininteligible la información que circulaba.

Si nos remontamos al tiempo de los egipcios o babilonios, vemos como utilizaban la escritura con jeroglíficos, o la cuneiforme que solo entendían algunos instruidos.

Probablemente, el primer caso claro de criptografía se dio en Esparta con el uso de la *escitala*. Los espartanos desarrollaron este método en su guerra contra Atenas. El historiador griego Plutarco en su obra "Vida de Lisandro", describe este sistema como:

"La escitala era un palo o bastón en el cual se enrollaba en espiral una tira de cuero. Sobre esa tira se escribía el mensaje en columnas paralelas al eje del palo. La tira desenrollada mostraba un texto sin relación aparente con el texto inicial, pero que podía leerse volviendo a enrollar la tira sobre un palo del mismo diámetro que el primero" [1].

La figura 1 muestra cómo sería una escitala.

Figura 1Escitala Espartana [2]

Posteriormente, en la época romana, apareció el *algoritmo de César*, que empleaba el emperador para enviar mensajes cifrados a sus legiones. Este sencillo método de

encriptación aplica el método de sustitución. Consistía en sustituir una letra por la que ocupaba tres lugares más adelante en el alfabeto:

En el libro de Jeremías de la Biblia, podemos ver el sistema criptográfico hebreo conocido como *atbash.* Las letras del mensaje de origen se sustituyen de la siguiente manera: si la letra original se encuentra en la línea superior se sustituye por la letra correspondiente de la línea inferior, y a la inversa.

Polybios, escritor griego, desarrolló un sistema un poco más sofisticado: Generó con las letras del alfabeto una tabla cuadrada de 5x5. El sistema de cifrado consistía en hacer corresponder a cada letra del alfabeto un par de letras que indicaban la fila y la columna donde se encontraba.

Estos sistemas que hemos analizado, suponen las bases de la criptografía clásica: la sustitución (el cifrado de *César*, el *atbash* y el de *Polybios*) y la transposición (escitala espartana).

En la Edad Media, el descubrimiento más importante fue el de *Al-Kindi* El sistema para resolver los enigmas criptográficos está descrito claramente en dos breves párrafos:

"Una manera de resolver un mensaje cifrado, si sabemos en qué lengua está escrito, es encontrar un texto llano escrito en la misma lengua, suficientemente largo, y luego contar cuantas veces aparece cada letra. A la letra que aparece con más frecuencia la llamamos "primera", a la siguiente en frecuencia la llamaremos "segunda"....y así hasta que hayamos cubierto todas las letras que aparecen en nuestro texto.

Luego observamos el texto cifrado que queremos resolver y clasificamos sus símbolos de la misma manera. Encontramos el símbolo que aparece con mayor frecuencia y lo sustituimos por la "primera" de nuestro texto, hacemos lo mismo con la "segunda" y así sucesivamente, hasta que hayamos cubierto todos los símbolos del criptograma que queremos resolver" [1].

Siguiendo estas pautas podemos hacer un análisis sobre la frecuencia de uso de las letras, obteniendo los resultados mostrados en la Figura 2.

Letras de alta frecuencia		Letras de frecuencia media		Letras de frecuencia baja	
Letra	Frecuencia %	Letra	Frecuencia %	Letra	Frecuencia %
e	16,78	r	4,94	y	1,54
a	11,96	u	4,80	q	1,53
o	8,69	i	4,15	b	0,92
l	8,37	t	3,31	h	0,89
s	7,88	c	2,92	El resto de las letras: g,f,v,w,j,z,x,k tienen frecuencias inferiores a 0.5% y se pueden considerar por tanto "raras":	
n	7,01	p	2,76		
d	6,87	m	2,12		

Figura 2 Frecuencia de uso letras en castellano [1]

Posteriormente han ido apareciendo otros sistemas, cada vez más sofisticados hasta que en la Segunda Guerra Mundial, los alemanes construyeron la conocida máquina *Enigma* (mostrada en la Figura 3) que fue contrarrestada por la *Colosus*, diseñada por los ingleses, que logró desenmascarar las claves de Enigma. El 1 de Junio de 1944 *Colosus* descifró un mensaje de Hitler donde se indicaba que esperaban el desembarco en Calais. Los americanos decidieron hacerlo, dos días más tarde, en Normandía y eso, parece ser, acortó la guerra en, al menos, dos años.

Figura 3 Máquina Enigma [3]

2.1.3.2 Criptografía Moderna

Apoyados en los grandes avances en el terreno de la electrónica digital, los computadores y la informática, ha habido un extraordinario desarrollo en el terreno de la Criptografía.

El principal problema de la encriptación ha sido siempre el de la ocultación de la clave, pues su descubrimiento permitiría la decodificación de todos los mensajes cifrados con dicha clave.

Las principales características de los sistemas modernos nos permiten establecer una serie de clasificaciones de los mismos

Según el tipo de cifrado, podemos clasificar los sistemas criptográficos en: Sistemas por transposición y sustitución, explicados anteriormente, y Sistemas por producto, consistentes en el cifrado empleando los métodos anteriores de manera combinada, es decir, cifrando con un sistema un texto que ya ha sido cifrado previamente con otro sistema o con el mismo.

A continuación se incluye otra clasificación de los cifrados según la naturaleza de la clave [4]:

- Simétricos: o basados en clave privada. Este tipo de sistemas utiliza la misma clave para cifrar que para descifrar el mensaje, o en su defecto, claves que deriven fácilmente una de la otra. En la Figura 4, se esquematiza la comunicación por medio de un sistema simétrico.

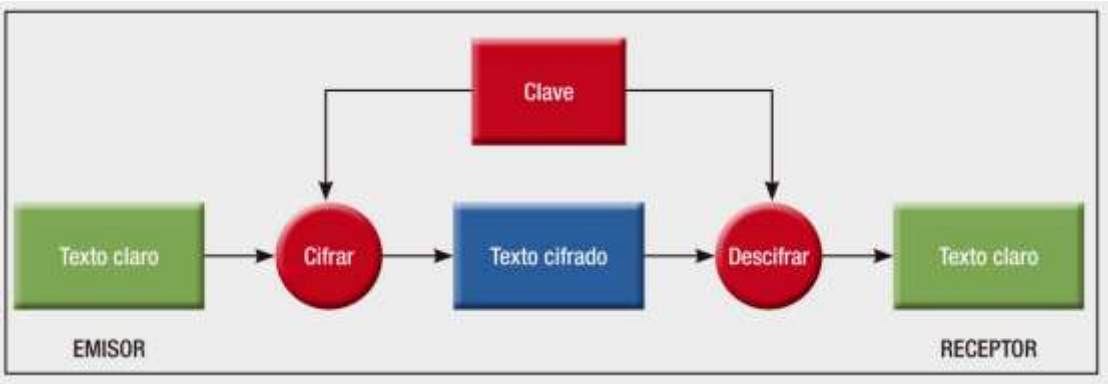

Figura 4 Sistema Criptográfico Simétrico [5]

- Asimétricos: o basados en clave pública. Estos sistemas en cambio, utilizan una clave muy diferente para cifrar/descifrar el mensaje, es por esto que en ocasiones se publican las claves en Internet para que cada cual pueda cifrar sus mensajes, siendo únicamente el diseñador el que pueda descifrarlos. Las dos claves pertenecen a la misma persona que ha enviado el mensaje. Una clave es *pública* y se puede entregar a cualquier persona, la otra clave es *privada* y el propietario debe guardarla de modo que nadie tenga acceso a ella. Además, los métodos criptográficos garantizan que esa pareja de claves sólo se puede generar una vez, de modo que se puede asumir que no es posible que dos personas hayan obtenido casualmente la misma pareja de claves. La Figura 5 muestra un esquema de un sistema asimétrico.

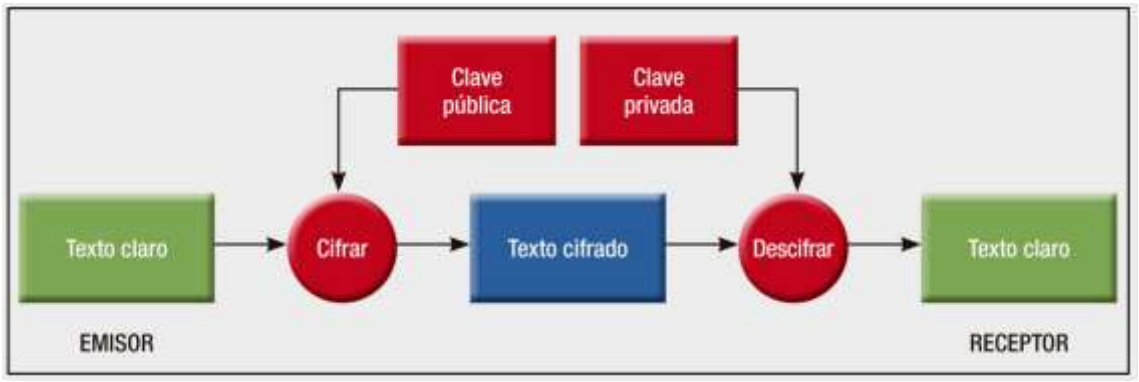

Figura 5 Sistema Criptográfico Asimétrico [5]

En el caso de un sistema simétrico, un buen sistema de cifrado pone toda la seguridad en la clave y ninguna en el algoritmo. No debería ser de ninguna ayuda para un atacante conocer el algoritmo que se está usando. Sólo si el atacante obtuviera la clave, le serviría conocer el algoritmo.

Dado que toda la seguridad está en la clave, es importante que sea muy difícil adivinar el tipo de clave. Esto quiere decir, que el abanico de claves posibles debe ser amplio. Actualmente, los ordenadores pueden descifrar claves con extrema rapidez, y ésta es la razón por la cual el tamaño de la clave es importante. El algoritmo de cifrado *DES* usa una clave de 56 bits, lo que significa que hay 2 elevado a 56 claves posibles (72.057.594.037.927.936 claves). Esto representa un número

muy alto de claves, pero un ordenador genérico puede comprobar el conjunto posible de claves en cuestión de días. Una máquina especializada puede hacerlo en horas. Algoritmos de cifrado de diseño más reciente como *3DES*, *Blowfish* e *IDEA* usan claves de 128 bits, lo que significa que existen 2 elevado a 128 claves posibles. Esto equivale a muchísimas más claves, y aun en el caso de que todas las máquinas del planeta estuvieran cooperando, tardarían más tiempo en encontrar la clave que la edad del universo.

El principal problema con los sistemas de cifrado simétrico no está ligado a su seguridad, sino al intercambio de claves. Una vez que el remitente y el destinatario hayan intercambiado las claves pueden usarlas para comunicarse con seguridad, pero ¿qué canal de comunicación que sea seguro han usado para transmitirse las claves? Sería mucho más fácil para un atacante intentar interceptar una clave que probar las posibles combinaciones del espacio de claves Es en este punto donde toman protagonismo los tipos de ataques para los que se realiza el estudio de este proyecto, los ataques de canal lateral que pretenden descubrir parámetros de la codificación a través de análisis físicos del sistema, es decir, a través de medidas de consumo, electromagnetismo, tiempo e incluso otras no tan evidentes como temperatura o sonido.

Otro problema es el número de claves que se necesitan. Si tenemos un número *n* de personas que necesitan comunicarse entre sí, se necesitan *n/2* claves para cada pareja de personas que tengan que comunicarse de modo privado. Esto puede funcionar con un grupo reducido de personas, pero sería imposible llevarlo a cabo con grupos más grandes.

En el caso de sistema asimétrico se cumplen los principios básicos de la criptografía moderna: *confidencialidad* del envío del mensaje, nadie salvo el destinatario puede descifrarlo. Si el propietario del par de claves usa su clave privada para cifrar el mensaje, cualquiera puede descifrarlo utilizando su clave pública. En este caso se consigue por tanto la *identificación* y *autentificación* del remitente, ya que se sabe que sólo pudo haber sido él quien empleó su clave privada (salvo que alguien se la hubiese podido robar).

La mayor ventaja de la criptografía asimétrica es que la distribución de claves es más fácil y segura ya que la clave que se distribuye es la pública manteniéndose la privada para el uso exclusivo del propietario, pero este sistema tiene bastantes desventajas:

- Para una misma longitud de clave y mensaje se necesita mayor tiempo de proceso.
- Las claves deben ser de mayor tamaño que las simétricas.
- El mensaje cifrado ocupa más espacio que el original.

Los nuevos sistemas de clave asimétrica basado en *curvas elípticas* tienen características menos costosas.

2.2 AES

En este apartado se describe el algoritmo criptográfico utilizado en el proyecto. Se comienza con su evolución e historia para pasar más adelante a una explicación de los fundamentos matemáticos en los que se basa y terminar con una descripción del funcionamiento tanto del cifrado como del descifrado.

2.2.1 Historia del AES

El primer estándar de cifrado se estableció en el año 1977. Se denominó DES (*Data Encryption Standard*) y había sido desarrollado por un equipo de *IBM* [6]. Sus características más importantes se pueden resumir en:

- El texto original se codifica en bloques de 64 bits, clave de 56 bits y 19 etapas diferentes.
- El descifrado se realiza con la misma clave y los pasos inversos.
- El inconveniente es que puede ser descifrado probando todas las combinaciones posibles, cosa que queda solucionada con el *Doble DES* (ejecuta el DES 2 veces con 3 claves distintas) y el *Triple Des* (2 claves y 3 etapas).

Este sistema fue atacado intensamente y finalmente mostró su debilidad. Mediante criptoanálisis se consiguió descifrarlo, pero fue con el diseño de una máquina, denominada *DES-CRACKER* que se logró descifrarlo por la fuerza bruta en un tiempo mínimo. Ante este descalabro, el *Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST)* convocó un proceso abierto para la creación de un nuevo y seguro algoritmo de cifrado, para ser utilizado de manera oficial por el Gobierno y por el sector privado.

En Septiembre de 1997 se dan los primeros pasos para la creación del Estándar de Cifrado Avanzado o *Advanced Encryption Standard* (AES) [7].

Los criterios mínimos que debían cumplir los estándares a evaluar serían, en resumen:

- El algoritmo será público
- Algoritmo simétrico cifrado en bloque
- La longitud mínima de la clave debe ser de 128 bits y podrá ser aumentada de acuerdo a las necesidades
- Debe ser implementado tanto en hardware como en software.

Los parámetros por los que serían juzgados estos estándares, además de las premisas anteriores, serían:

- Seguridad
- Eficiencia computacional
- Requisitos de memoria
- Simplicidad de diseño
- Flexibilidad

Se requería, además de lo anterior, que estos algoritmos tuvieran una longitud de bloque de, al menos, 128 bits y una clave de 128, 192 y 256 bits.

Cualquier persona, institución u organismo podía participar en este concurso presentando algoritmos o también presentando críticas,

El *NIST* propuso realizar, con los algoritmos presentados, dos rondas de selección. En la primera, quedarían seleccionados los cinco algoritmos que obtuvieran mejor puntuación y en de la segunda saldría el algoritmo o algoritmos ganadores. Se convocaron tres conferencias en diferentes lugares del mundo donde los algoritmos presentados fueron probados por quien así lo quiso, porque era condición indispensable que todos los algoritmos y criterios de diseño estuvieran disponibles de forma pública y abierta. Esto da una idea de lo exhaustivo que fue el proceso de elección donde los participantes no solo presentaron sus trabajos, sino que analizaron las debilidades de sus oponentes.

A continuación se describen las diferentes rondas en las que consistió el concurso [8].

Primera ronda

El 20 de agosto de 1998 el *NIST* anunció los 15 algoritmos admitidos en la primera conferencia de candidatos a AES:

- CAST-256 (*Entrust Technologies, Inc*.)
- CRYPTON (*Future Systems, Inc*.)
- DEAL (*Richard Outerbridge, Lars Knudsen*)
- DFC (*CNRS – Centre National pour la Recherche Scientifique – Ecole Normale Superieure*)
- E2 (*NTT – Nippon Telegraph and Telephone Corporation*)
- FROG (*TecApro International, S.A.*)
- HPC (*Rich Schroeppel*)
- LOKI97 (*Lawrie Brown, Josef Pieprzyk, Jennifer Seberry*)
- MAGENTA (*Deutsche Telekom AG*)
- MARS (*IBM*)
- RC6 (*RSA Laboratories*)
- RIJNDAEL (*John Daemen, Vincent Rijmen*)
- SAFER+ (*Cylink Corporation*)
- SERPENT (*Ross Anderson, Eli Biham, Lars Knudsen*)
- TWOFISH (*Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson*)

Segunda ronda

La segunda conferencia AES tuvo lugar en marzo de 1999 donde se discutieron las numerosas pruebas y análisis a los que fueron sometidos los candidatos por la comunidad criptográfica internacional. Se admitieron comentarios hasta el 15 de abril. El NIST decidió en agosto de 1999 cuales serían los 5 finalistas:

- MARS (IBM Nevenki Zunic)
- RC6 (RSA Laboratories. Rivest, M. Robshaw, Sidney, Yin)
- RIJNDAEL (Joan Daemen, Vicent Rijmen)
- SERPENT R. Anderson, E.Biham, L. Knudsen)
- TWOFISH (B Schneier, J. Kelsey, D. Whaiting, D. Wagner, C. May, N. Ferguson)

Estos algoritmos fueron sometidos a una segunda revisión, más exhaustiva, que duró hasta el 15 de mayo de 2000. Durante este periodo el NIST admitió análisis de los algoritmos finalistas.

Tercera ronda

Durante los días 13 y 14 de abril de 2000, en Nueva York, tuvo lugar la tercera conferencia AES donde se discutieron las últimas aportaciones de evaluación y criptoanálisis de los algoritmos finalistas. En ella estuvieron presentes los desarrolladores de los algoritmos finalistas.

El 15 de mayo de 2000 finalizó el periodo público de análisis. El NIST estudió toda la información disponible para decidir cuál sería el algoritmo ganador. El 2 de octubre de 2000 se votó cual sería el algoritmo que finalmente ganaría el concurso. El resultado fue el siguiente:

- RIJNDAEL: 86 votos
- SERPENT: 59 votos
- TWOFISH: 31 votos
- RC6: 23 votos
- MARS: 13 votos

El algoritmo *Rijndael* ganó el concurso y en noviembre de 2001 se publicó en FIPS 197 donde se asumía oficialmente.

Fue creado por dos ingenieros belgas: Vincent Rijmen, matemático de la *Universidad Católica de Lovaina* y Joan Daemen de la misma universidad. Ambos compitieron, sin grandes apoyos, contra equipos de multinacionales de la categoría de Deutsche Telekom, IBM o famosos criptógrafos como Bruce Schneier (*Twofish*) o Ronald Rivest (*RC6*). El nombre de *Rijndael* viene de la mezcla de los apellidos de sus creadores.

Para probar la característica más importante de los algoritmos, su seguridad, tuvieron que soportar diversos ataques lanzados, tanto por sus contrincantes, como por cualquier persona o entidad que presentara su trabajo al *Instituto Nacional de Estándares y Tecnología de EEUU*, que dio como resultado:

- *RIJNDAEL*: Demostró tener un margen de seguridad adecuado, aunque era de los más bajos entre los finalistas. Debido a que su estructura matemática era bastante simple, facilitó su análisis de seguridad en el tiempo marcado por el NIST.
- *SERPENT*: Margen de seguridad alto. Debido a que su estructura matemática era bastante simple, también facilitó su análisis de seguridad en el tiempo marcado por el NIST.
- *TWOFISH*: Margen de seguridad también alto. Su complejidad hizo difícil su análisis de seguridad en el tiempo marcado por el NIST.
- *RC6*: Demostró tener un margen de seguridad adecuado pero inferior al de los otros contrincantes. Debido a que su estructura matemática era bastante simple, también facilitó su análisis de seguridad en el tiempo marcado por el NIST.
- MARS: Margen de seguridad elevado. Su complejidad hizo difícil su análisis de seguridad en el tiempo marcado por el NIST.

En los casos de *MARS, RC6* y *SERPENT*, la velocidad con que ejecutaba cada algoritmo no dependía excesivamente del tamaño de la clave, mientras que Rijndael

y *Twofish* eran más lentos según aumentaba el tamaño de la clave, aspectos que compensaban aumentando su seguridad [9].

2.2.2 El Algoritmo Rijndael

El algoritmo Rijndael fue una actualización de un diseño anterior de Daemen y Rijmen, el *Square* que fue a su vez un desarrollo de *Shark*.

Al contrario que su predecesor *DES*, *Rijndael* es una red de sustitución-permutación, no una red de *Feistel*. AES es rápido tanto en software como en hardware, es relativamente fácil de implementar, y requiere poca memoria. Como nuevo estándar de cifrado, se está utilizando actualmente a gran escala.

Para entender mejor su funcionamiento, vamos a esbozar sus principios matemáticos [7].

2.2.2.1 Campos de Galois GF(2^8). Cuerpos Finitos.

En este algoritmo todos los bytes se interpretan como elementos de un cuerpo finito. Esto es, mediante *Campos de Galois*, en inglés *Galois Fields: GF(k).*

El interés de estos campos radica en que existe un inverso aditivo y multiplicativo que permite cifrar y descifrar en el mismo cuerpo Z_k, lo que supone prescindir de los problemas de redondeo o truncamiento que hubieran aparecido si trabajáramos en la aritmética real.

Interesa utilizar una aritmética en módulo "p" sobre los polinomios de grado "m", siendo p un número primo, dando lugar a un campo de *Galois* GF (p^m), donde sus elementos se representan como polinomios con coeficientes en Z_p, de grado menor que m. Estos polinomios serán del tipo:

$$GF(p^m) = \{ \lambda_0 + \lambda_1 x + \lambda_2 x^2 + \lambda_3 x^3 + \dots \lambda_{m-1} x^{m-1} \}$$

Donde $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \dots \lambda_{m-1} \in Z_p$

Cada elemento de GF(p^m) es un resto módulo p(x), donde p(x) es un polinomio irreducible de grado "m", esto significa que no puede ser factorizado en polinomios de grado menor que "m".

En nuestro caso, nos interesaran los campos del tipo GF(2^m) porque los coeficientes serán los restos del módulo 2, es decir 0 y 1, lo que permite una representación binaria. Por lo tanto, cada elemento del campo se representa con m bits y el número de elementos será 2^m. Por ejemplo, para el campo GF(2^3), sus elementos son los restos de un polinomio de grado m-1:

$0, 1, x, x+1, x^2+1, x^2+x, x^2+x+1.$

En el caso del algoritmo *Rijndael*, se definen operaciones a nivel de byte, por lo que el campo será GF(2^8). Teniendo en cuenta que un byte se compone de los bits: $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$ si lo consideramos como un polinomio en {0, 1} obtendremos el polinomio:

$$b_7x^7+ b_6x^6+ b_5x^5+ b_4x^4+ b_3x^3+ b_2x^2+ b_1x^1+ b_0x^0$$

Por ejemplo un byte hexadecimal con valor "35", corresponde con el polinomio:

$$x^5+x^4+x^2+1$$

2.2.2.2 Suma en GF (2^8)

Las operaciones de suma y resta se hacen con una Or-Exclusiva ya que si los coeficientes son iguales, el resultado será un cero y si son distintos un uno, debido a que hemos considerado que las operaciones matemáticas sobre coeficientes se hacen en módulo p, por lo que en nuestro caso, GF (2^m), los resultados resultan de la suma de los coeficientes módulo 2.

Entonces, para sumar dos polinomios, se aplicará la operación Or-Exclusiva a cada elemento de los mismos, dos a dos.

Por ejemplo: Si sumamos dos polinomios en GF (2^8), veamos el resultado:

$$P_1 = 35_{16} = 00110101 = x^5+ x^4+ x^2+ 1$$

$P_2 = 46_{16} = 01000110 = x^6+x^2+x$

$P_1 + P_2 = x^6+x^5+ x^4+ 2x^2+x+ 1_{mod\ 2} = x^6+x^5+ x^4+x+ 1_{mod\ 2} = 01110011 = 73_{16}$

Se observa que se obtienen los mismos resultados aplicando la operación lógica OR-Exclusiva con estos mismos polinomios:

$00110101 \oplus 01000110 = 01110011 = 73_{16}$

La operación Or-Exclusiva se describe acorde a su tabla de verdad representada en la Figura 6.

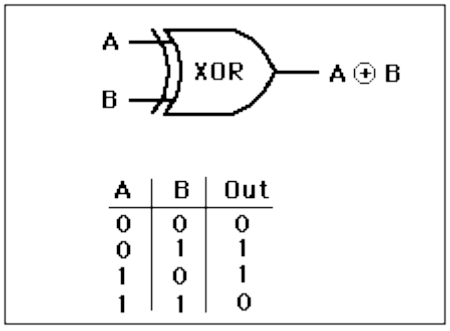

Figura 6 Operación lógica Or-Exclusiva

2.2.2.3 Multiplicación en GF(2^8)

En la multiplicación de polinomios en GF (2^m), es posible que el resultado contenga elementos que estén fuera del cuerpo del polinomio (potencias iguales o mayores que "m") por lo que deberemos reducir los exponentes mediante un polinomio p(x) necesariamente irreducible y grado "m" (es irreducible porque sus únicos divisores son el 1 y el mismo polinomio).

Para GF(2^8) la multiplicación de polinomios se realiza modulo con un polinomio irreducible de grado 8. Este polinomio irreducible se representa por m(x) El polinomio irreducible utilizado en el algoritmo Rijndael es:

$m(x) = x^8+x^4+x^3+x+1$

Ejemplo de una multiplicación de polinomios: '57' • '83' = 'C1', esto es así porque:

$A = 57_{16} = 0101\ 0111_2 = x^6 + x^4 + x^2 + x + 1$

$B = 83_{16} = 1000\ 0011_2 = x^7 + x + 1$

$A*B = (x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + 2x^7 + x^6 + x^5 + x^4 + x^3 + 2x^2 + 2x + 1 \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod x^8 + x^4 + x^3 + x + 1 = x^7 + x^6 + 1 =$

$= 1100\ 0001 = C1_{16}$ [10]

(2-1)

Otra forma de calcularlo sería razonando qué resultado de la multiplicación hay que reducirlo por m(x) para cada valor de x que esté fuera del cuerpo de 8 bits:

Sea $m(x) = x^8 + x^4 + x^3 + x + 1 \rightarrow \underline{x^8 = x^4 + x^3 + x + 1}$

$A*B \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$x^{13} = x^5 \bullet x^8 = x^5 \bullet (x^4 + x^3 + x + 1) = x^9 + x^8 + x^6 + x^5 = x \bullet x^8 + x^8 + x^6 + x^5 =$

$x \bullet (x^4 + x^3 + x + 1) + (x^4 + x^3 + x + 1) + x^6 + x^5 =$

$(x^5 + x^4 + x^2 + x) + (x^4 + x^3 + x + 1) + x^6 + x^5$

$x^{13} = x^6 + x^3 + x^2 +$

$x^{11} = x^3 \bullet x^8 = x^3 \bullet (x^4 + x^3 + x + 1) = \underline{x^7 + x^6 + x^4 + x^3}$

$x^9 = x \bullet x^8 = x \bullet (x^4 + x^3 + x + 1) = \underline{x^5 + x^4 + x^2 + x}$

Por tanto, el resultado de multiplicar los polinomios, se obtiene sustituyendo:

$A \bullet B \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$A \bullet B \bmod 2 = (x^6 + x^3 + x^2 + 1) + (x^7 + x^6 + x^4 + x^3) + (x^5 + x^4 + x^2 + x) + (x^4 + x^3 + x + 1) + x^6 + x^5 + x^4 + x^3 + 1 \bmod 2$

$A \bullet B \bmod 2 = x^7 + x^6 + 1 = 1100\ 0001 = C1_{16}$

Como era lógico esperar, el polinomio resultante tendrá grado menor que 8. La multiplicación de polinomios es asociativa y su elemento neutro es el "01". Para cualquier polinomio binario b(x) de grado menor que 8, se puede aplicar el algoritmo extendido de Euclides para calcular un polinomio inverso de b(x). En este caso se habla de "inversa multiplicativa": a(x) es un polinomio inverso de b(x) si:

$$a(x) \bullet b(x) \bmod m(x) = 1 \quad \text{ó} \quad b^{-1}(x) = a(x) \bmod m(x)$$

Es decir, a(x) es la inversa multiplicativa de b(x). El polinomio extendido de Euclides se puede ver como:

$$b(x)a(x)+m(x)c(x)=1$$

2.2.2.4 Multiplicación por x

A continuación se va a analizar un caso interesante, que es la multiplicación de un polinomio por x. Si multiplicamos un polinomio b(x) por x tenemos:

$$b(x)=b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

$$b(x) \bullet x = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

Una vez tenemos este resultado se debe realizar la reducción modulo m(x). Si b7=0 el resultado es el mismo polinomio. Si b_7=1, m(x) debe anular el valor de x^8.

En general, para utilizar este tipo de multiplicación los autores del algoritmo AES proponen una función denominada *xtime* que simplifica la multiplicación de un polinomio por potencias de x [10]. Este hecho es gracias a que la función *xtime* se puede ejecutar de forma reiterativa. La función *xtime* consiste en aplicar un desplazamiento a la izquierda al valor que representa el polinomio y una operación or-exclusiva con el valor 0x11B (0x11B=000100011011= m(x)=x8+x4+x3+x+1) cuando el resultado de la multiplicación debe ser reducido módulo m(x). Esta función se puede programar fácilmente de la siguiente manera: [7]

```
Int xtime (int valor)

{valor=valor<<1;
```

```
if(valor&0x100) valor^=0x11B;
return valor;}
```

Ejecutar una vez la función xtime equivale a multiplicar el polinomio representado por su "valor" por x, es decir el polinomio • '02'.

La importancia de esta función recae en su uso reiterado para calcular multiplicaciones de polinomios. Por ejemplo la multiplicación de los siguientes polinomios:

'57' • '13'= $(x^6+x^4+x^2+x+1) \bullet (x^4+x+1)$

Se puede ver como la multiplicación de '57' por diversas potencias de x, aplicando la propiedad asociativa. Según esto, el polinomio '13' se puede descomponer en potencias de x de la siguiente forma:

'13'='01''⊕02'⊕ '10' → $(x^4+x+1)= 1 \oplus x \oplus x^4$

Por lo tanto:

'57' • '13'= '57' • ('01' ⊕ '02'⊕ '10') = ='57'⊕ ('57•02') ⊕ ('57•10')

Que resolviéndolo da:

'57'• **'02'** = $(x^6+x^4+x^2+x+1) \bullet x$ = xtime(57) = 'AE'

'57'• '04'= $(x^6+x^4+x^2+x+1) \bullet x^2$ = xtime(AE) ='47**'**

'57**'**• '08**'**= $(x^6+x^4+x^2+x+1) \bullet x^3$ = xtime(47) ='8E**'**

'57**'**• '10**'**= $(x^6+x^4+x^2+x+1) \bullet x^4$ = xtime(8E) =**'**07**'**

El cálculo de las cuatro llamadas a la función xtime se observa en la siguiente tabla:

Operaciones mediante la función xtime.

Finalmente se calcula:

'57'•'13'='57'•('01'⊕'02'⊕'10')=

'57'⊕('57'•'02')⊕('57•'10')='57'⊕'AE'⊕'07'='FE

Este procedimiento es interesante ya que las multiplicaciones que se realizan en el algoritmo se pueden hacer utilizando esta función. Su uso se centra en la función MixColumn del algoritmo con polinomios representados por '01', '02', '03', '09', '0b', '0d' y '0e'

Consideraciones:

- Multiplicar un polinomio por '01' es igual al mismo polinomio.
- Multiplicar un polinomio por '02' consiste en aplicar la función xtime al polinomio.
- Multiplicar un polinomio A por '03' es igual a:

  $(A \bullet \text{'02'}) \oplus A = \text{xtime}(A) \oplus A.$

- Multiplicar un polinomio A por '09' es igual a:

  $(A \bullet \text{'08'}) \oplus A = \text{xtime}(\text{xtime}(\text{xtime}(A))) \oplus A$

- Multiplicar un polinomio A por '0b' es igual a:

  $(A \bullet \text{'08'}) \oplus A = \text{xtime}(\text{xtime}(\text{xtime}(A))) \oplus A$

- Multiplicar un polinomio A por '0e' es igual a:

  $(A \bullet \text{'02'}) \oplus (A \bullet \text{'04'}) \oplus (A \bullet \text{'08'}) =$

  $\text{xtime}(A) \oplus \text{xtime}(\text{xtime}(A)) \oplus \text{xtime}(\text{xtime}(\text{xtime}(A)))$

- Multiplicar un polinomio A por '0d' es igual a:

  $(A \bullet \text{'04'}) \oplus (A \bullet \text{'08'}) \oplus A =$

  $\text{xtime}(\text{xtime}(A)) \oplus \text{xtime}(\text{xtime}(\text{xtime}(A))) \oplus A$

2.2.2.5 Representación de palabras en el campo GF(2)

Una palabra es un conjunto de bytes. El algoritmo AES usa palabras de 4 bytes (32 bits), considerándolas como un polinomio de grado menor que cuatro con

coeficientes que son a su vez polinomios en GF(2^8). De esta manera queda definida una palabra con esta expresión:

$$A(x) = a_3x^3 + a_2x^2 + a_1x + 1$$

La suma de palabras se realiza mediante operaciones OR-Exclusivas byte a byte, análogamente a lo realizado en GF (2^8). Esta operación no contrae problemas de desbordamiento puesto que como resultado siempre va a darnos otro polinomio de grado menor que 4.

En la multiplicación, al igual que en GF (2^8), sí que nos podemos encontrar con el problema del desbordamiento. El producto de dos palabras de 4 bytes puede no ser representable por una palabra de 4 bytes, o sea, mediante un polinomio de grado menor que 4. Esta operación por tanto no es una operación interna en GF(2^8).

Por esto, se adopta la solución de expresar el resultado con módulo un polinomio de grado 4. Los autores del algoritmo eligieron el polinomio M(x) como el módulo de la operación ya que todas las multiplicaciones que se pueden realizar en el algoritmo Rijndael se realizan con polinomios que poseen inverso [7].

$$M(x) = x^4 + 1$$

2.2.2.6 Consideraciones previas de diseño del algoritmo Rijndael

Joan Daemen y *Vincent Rijmen* diseñaron el algoritmo Rijndael basándose en tres criterios fundamentales:

- Resistencia contra la totalidad de los ataques conocidos.
- Velocidad, código compacto y operativo en multitud de plataformas distintas.
- Simplicidad de diseño.

Una de las principales diferencias entre Rijndael y el resto de los algoritmos de cifrado simétricos existentes, radica en que Rijndael no tiene una estructura interna tipo

Feistel. En una estructura tipo Feistel, en cada una de las operaciones de cada vuelta, todos los bits sufren una permutación pero la mayoría de los bits no varían su valor, es decir, llegan a su nueva posición sin ningún cambio en su valor.

Rijndael no tiene una estructura tipo Feistel ya que trata todos los bits por vuelta. En Rijndael, cada vuelta está compuesta por tres transformaciones invertibles y distintas entre sí llamadas *layers* (capas). Estas transformaciones están basadas en el principio de diseño "*Wide Trail*" que da resistencia al algoritmo frente a ataques de tipo lineal y diferencial. Cada capa tiene su propia función específica dentro de esta estrategia:

- Capa de mezcla lineal: Garantiza una gran difusión de la información a través de la aplicación de varias rondas.
- Capa no lineal: Consiste en la aplicación paralela de la S-Box para conseguir unas óptimas propiedades de no linealidad
- Capa de adición de clave: Se mezcla el estado intermedio con la correspondiente subclave de ronda mediante una simple operación OR-Exclusiva.

Además, antes de la primera ronda del algoritmo, se aplica la capa de adición de clave. Esto se realiza para impedir que se ataquen directamente las claves. Por el contrario, cualquier capa que apliquemos tras la última adición de clave o antes de la primera, en los ataques basados en textos en claro conocidos, puede ser descubierta sin conocer la clave y esto sería un fallo en la seguridad del algoritmo, como se puede apreciar en la permutación inicial y final del algoritmo DES. Esta técnica no es nueva, puesto que ya aparece en algoritmos como *IDEA* o *Blowfish*.

En la última ronda, con el objetivo de que el algoritmo de cifrado y el de descifrado sean lo más similares posible, se aplica una capa de mezcla lineal distinta a la de las rondas anteriores. No obstante, esta técnica también aplicada en algoritmos como DES, no reduce la seguridad del algoritmo.

2.2.3 Descripción del algoritmo AES para Encriptación

El algoritmo Rijndael es un sistema simétrico de cifrado por bloques, por tanto utiliza la misma clave para el proceso de cifrado como para el proceso de descifrado. Su diseño permite la utilización de claves de sistema con longitud variable siempre que sea múltiplo de 4 bytes. La longitud de las claves utilizadas por defecto son 128 (AES-128), 192 (AES- 192) y 256 (AES-256) bits. De la misma manera el algoritmo permite la utilización de bloques de información con un tamaño variable siempre que sea múltiplo de 4 bytes, siendo el tamaño mínimo recomendado de 128 bits (el tamaño mínimo de 16 bytes).

Estrictamente hablando, AES no es precisamente Rijndael (aunque en la práctica se les llama de manera indistinta) ya que Rijndael permite un mayor rango de tamaño de bloques y longitud de claves; AES tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 o 256 bits, mientras que Rijndael puede ser especificado por una clave que sea múltiplo de 32 bits, con un mínimo de 128 bits y un máximo de 256 bits.

Este algoritmo opera a nivel de byte, interpretando éstos como elementos de un cuerpo de *Galois* GF(28), y a nivel de registros de 32 bits, considerándolos como polinomios de grado menor que 4 con coeficientes que son a su vez polinomios en GF(28)

La estructura del algoritmo Rijndael está formada por un conjunto de "rondas", entendiendo por "rondas" un conjunto de reiteraciones de 4 funciones matemáticas diferentes e invertibles. Este proceso se puede expresar de forma gráfica como en la Figura 7.

Por tanto, el algoritmo se basa en aplicar un número de rondas determinado a una información en claro para producir una información cifrada. La información generada por cada función es un resultado intermedio, que se conoce como Estado, o si se prefiere Estado Intermedio.

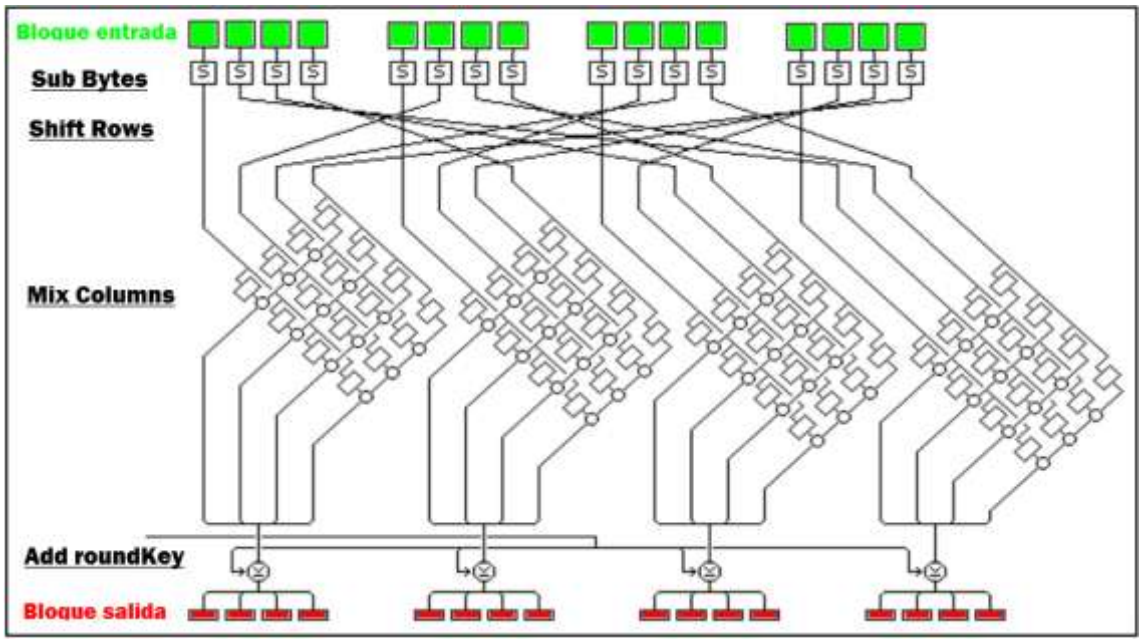

Figura 7 Representación gráfica de una ronda AES

Tanto para el proceso de cifrado como para el de descifrado, en cada una de las rondas se utiliza una subclave generada anteriormente mediante la función *KeyExpansion* a partir de la clave de cifrado. La función *KeyExpansion* devuelve una subclave más que el número de rondas que tiene el algoritmo. Esta subclave extra se utiliza antes de realizar todo el proceso de cifrado o descifrado a la matriz de estado con el objetivo de imposibilitar el criptoanálisis.

En todas las rondas, menos la última, el algoritmo realiza una serie de operaciones que van modificando la matriz de estado aportando confusión o difusión al mensaje a cifrar.

A continuación se enuncian las operaciones que se aplican y posteriormente se describirán más detalladamente:

- Función *SubBytes*. Aporta confusión al proceso al realizar una sustitución byte a byte con propiedades óptimas de no linealidad.
- Función *ShiftRows*. Introduce difusión de información a la ronda mediante la rotación de las filas del estado.
- Función *MixColumns*. Permite un alto nivel de difusión mezclando las columnas entre sí.
- Función *AddRoundKey*. Introduce un grado de confusión que depende de la subclave de ronda.

El proceso de cifrado consiste en aplicar cuatro funciones matemáticas invertibles a la información a cifrar. Estas funciones se repiten en cada vuelta.

La información a cifrar se va colocando en la matriz de estado, sobre la cual se realizarán las transformaciones. El proceso completo, consistente en varias rondas se esquematiza en la Figura 8.

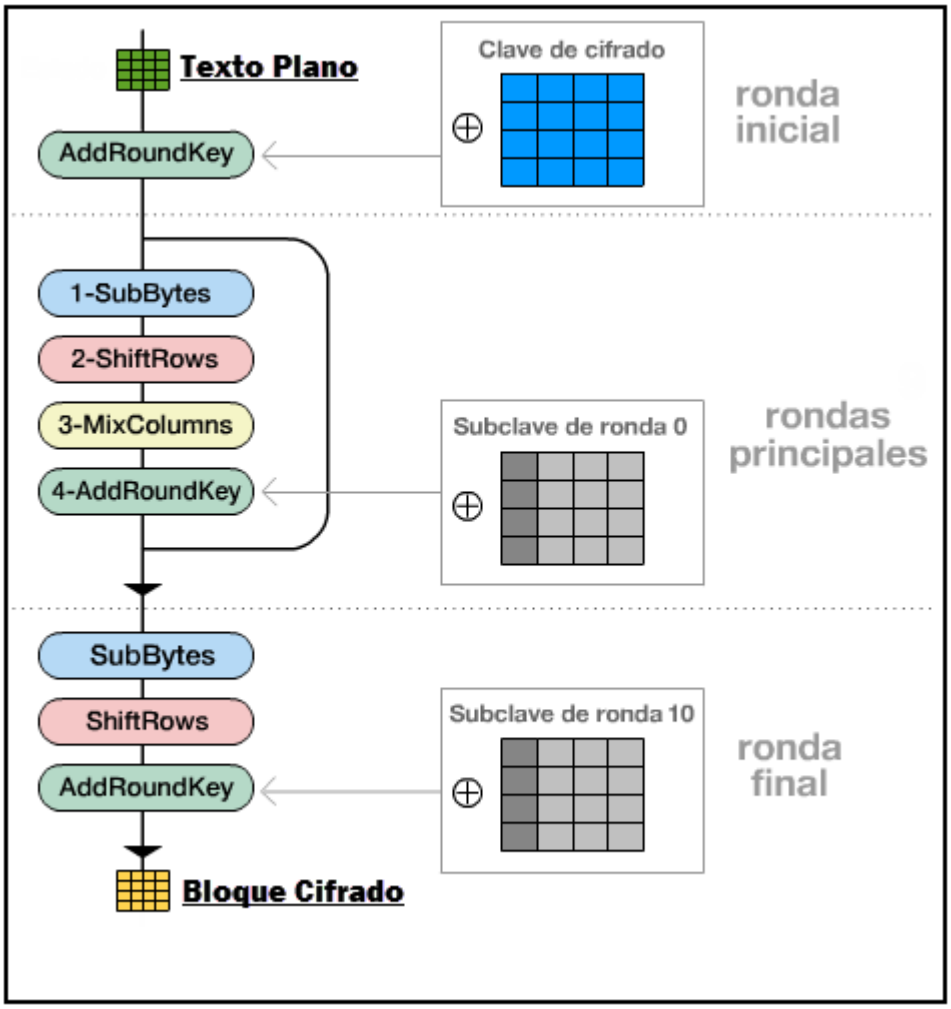

Figura 8 Diagrama de flujo completo de algoritmo AES [11]

Antes de realizar la primera ronda, se aplica la primera transformación a la matriz de estado. Se trata de la función *AddRoundKey* y consiste en una operación OR-Exclusiva entre la primera subclave y la matriz de Estado. A continuación, a la matriz de estado resultante se le aplican cuatro transformaciones invertibles (*SubBytes*, *ShiftRows*, *MixColumns* y *AddRoundKey*), repitiéndose estas cuatro

transformaciones hasta llegar a la penúltima ronda, en lo que se conoce como la vuelta estándar.

Finalmente se le aplica una última ronda a la matriz de estado resultante de las anteriores vueltas, aplicando solamente las funciones *SubBytes*, *ShiftRows* y *AddRoundKey* en este orden. El resultado de la vuelta final produce el bloque cifrado deseado.

A continuación describiremos con más detenimiento cada una de las funciones que se realizan durante el proceso de cifrado.

2.2.3.1 Función SubBytes

La función *SubBytes* aporta una capa no lineal diseñada específicamente para ofrecer resistencia frente a ataques como el criptoanálisis diferencial, el criptoanálisis lineal, ataques de interpolación, etc. Los criterios para diseñarla fueron:

- Que sea invertible
- Minimizar la relación lineal entre bits de entrada y bits de salida.
- Complejidad mediante expresiones algebraicas en GF(2^8)
- Sencillez de diseño.

Existen diversas formas para construir la *S-Box*. Daemen y Rijmen, en un primer momento, definieron la tabla de sustitución mediante el cálculo de inversas multiplicativas en GF(2^8), lo cual permite propiedades de no-linealidad. Sin embargo, la sencillez algebraica de este diseño podría permitir manipulaciones algebraicas que permitieran ataques. Para evitarlo se añadió una transformación afín invertible adicional.

La transformación *SubBytes* consiste en una sustitución no lineal que se aplica a cada byte de la matriz de Estado de forma independiente, generando un nuevo byte.

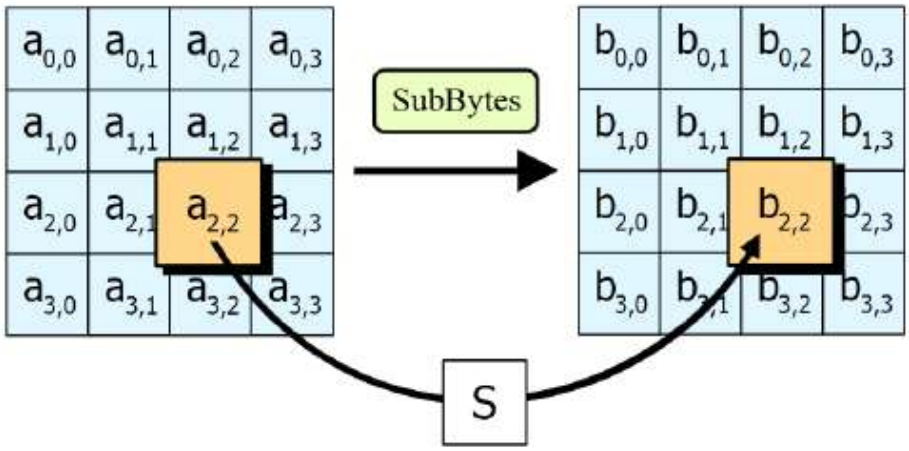

Figura 9 Transformación SubBytes [12]

Esta transformación consiste en la sustitución de cada byte por el resultado de aplicarle la tabla de sustitución S-Box. Esta tabla lógicamente es invertible y se construye mediante dos transformaciones:

- 1ª Transformación. Cada byte es considerado como un elemento en GF(2^8) que genera el polinomio irreducible m(x)= $x^8 + x^4 + x^2 + x + 1$, siendo sustituido por su inversa multiplicativa. El valor cero queda inalterado, ya que no tiene inversa.
- 2ª Transformación. Al resultado de la 1ª transformación se le aplica la transformación afín en GF(2^8) definida en la Figura 10, siendo x0, x1, x2, x3, x4, x5, x6 y x7 los bits del byte resultante de la 1ª transformación, e y0, y1, y2, y3, y4, y5, y6 e y7 los bits del resultado final de la transformación ByteSub.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figura 10 Matriz para 2ª Transformación en SubBytes

Por ejemplo, si el byte al que se le aplica la función SubBytes es "A=11001011", deberíamos calcular su inversa multiplicativa. Cada byte en Rijndael se representa como un polinomio a(x), calcular la inversa multiplicativa consiste en buscar un polinomio b(x) (que es único) que multiplicado por a(x) modulo m(x) es igual a 1, es decir:

$$a(x) \bullet b(x) \bmod x^8 + x^4 + x^3 + x + 1 = 1$$

El polinomio buscado en este caso es b(x)= x^2 que tiene una representación binaria de "B=00000100" Se dice entonces que B es la inversa en GF(2^8) de A. Una vez que tenemos el resultado de la primera transformación, debemos aplicarle la transformación afín definida anteriormente, obteniendo el valor final de "Y=00011111". Luego el byte "A=11001011" se convierte en Y al aplicar la función SubBytes.

Utilizando estas dos transformaciones para todos los valores posibles de entrada (256 valores ya que se trabaja con un byte) se calcula una tabla de sustitución denominada S-Box útil para el proceso de cifrado.

Gracias a esta tabla, aplicar la función SubBytes resulta trivial ya que consiste en dividir el byte de la matriz de Estado en dos partes de 4 bits. Los 4 bits más significativos, denominados por x (toma valores de 0 a 15) y actúan de fila en la tabla mientras que los 4 bits menos significativos, denominados por y (toma valores de 0 a 15) actúan de columna. El valor para esa fila y columna en la tabla es resultado de aplicar S-BOX a un byte.

Siguiendo con el ejemplo anterior, si tenemos el byte "A=11001011" y le aplicamos la función SubBytes el resultado será (x=1100 [fila c] y=1011 [columna b]) "Y=0x1F" que en binario equivale a "Y=00011111", valor que es idéntico al calculado previamente. El la Figura 11 se muestra la tabla de sustitución en notación hexadecimal.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 11 Tabla de sustitución S-BOX [13]

Para el proceso de descifrado es necesario calcular la función inversa de *SubBytes*. Esta función inversa consiste en calcular una tabla inversa a la utilizada en el proceso de cifrado. En la Figura 12 se muestra la tabla inversa S-Box, también en notación hexadecimal.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figura 12 Tabla de sustitución inversa S-BOX [13]

2.2.3.2 Función *ShiftRow*

El número de desplazamientos que se realizan en esta transformación, fue elegido para ofrecer la máxima resistencia contra ataques conocidos entre los que destacan

ataques *diferenciales* como el *truncated differentials* o ataques integrales como el *ataque Square*. Se demostró que para ataques de este tipo, realizando ciertos desplazamientos, el algoritmo presentaba mayor resistencia. Basándose en este principio, se escogieron los mejores desplazamientos contra estos ataques [7].

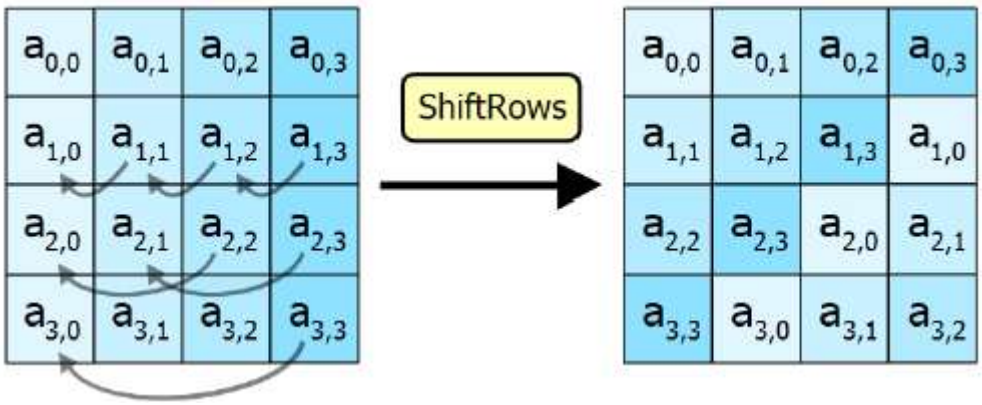

Figura 13 Función ShiftRows [12]

Esta transformación consiste en rotar a la izquierda las filas que conforman la matriz de Estado, es decir, rotar los bytes de las filas de la matriz de estado resultante de la transformación anterior (función *SubBytes*) a la izquierda.

La estructura del algoritmo fue diseñada para permitir cualquier tamaño de bloque que sea múltiplo de 4 bytes, con un número mínimo de 16 bytes. Las funciones *AddRoundKey*, *ByteSub* y *MixColumn* son independientes del tamaño de bloque. Sin embargo la transformación *ShiftRow* si depende de la longitud del bloque, siendo necesario definir valores C1, C2 y C3 diferentes para distintos bloques. Teniendo en cuenta esto, se han facilitado valores adicionales de C1, C2 y C3 para otras longitudes comunes de bloques (siendo Ci el número de rotaciones de las filas):

Tamaño de bloque	C1	C2	C3
160 bits (Nb=5)	1	2	3
224 bits (N b = 7)	1	2	4

Figura 14 Valores Ci para diferentes tamaños de bloque

La función inversa de *ShiftRow* que permite invertir esta transformación, consiste simplemente en rotar a la derecha los bytes de las filas de la matriz de Estado actual. Para ello se desplaza el mismo número de posiciones C¡ que se desplazaron para cifrar.

2.2.3.3 Función *MixColumn*

La función *MixColumn* ha sido seleccionada como transformación lineal de 4 en 4 bytes, teniendo en cuenta los siguientes criterios:

- Que sea invertible
- Linealidad en GF(2^8)
- Propiedades de alta difusión.
- Velocidad en procesadores de 8 bits.
- Simetría.
- Sencillez de descripción.

La elección de los coeficientes y los polinomios permiten una gran difusión entre los bytes resultantes de la transformación.

En la Figura 15 se muestra la transformación que realiza la función MixColumns.

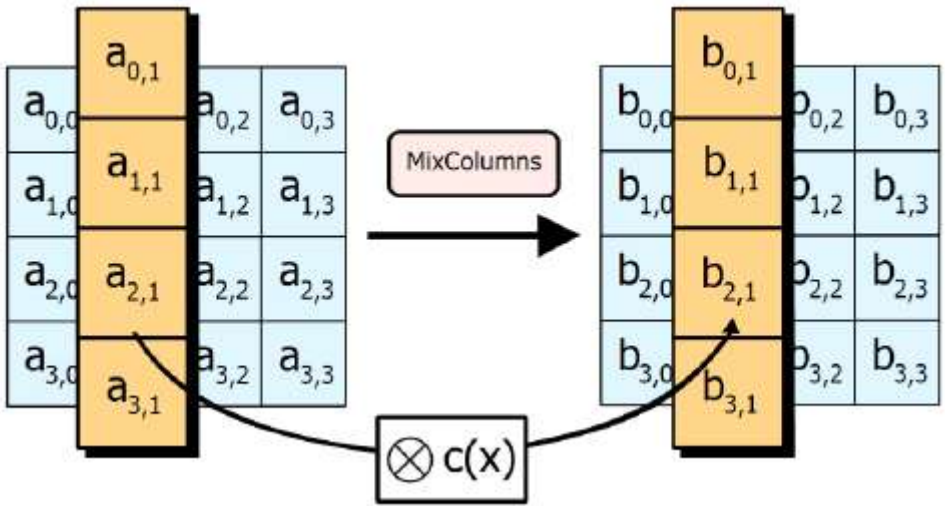

Figura 15 Función Mixcolumns [12]

La transformación MixColumn actúa sobre los bytes de una misma columna de la matriz de Estado que tiene a la entrada. En esencia esta función permite una mezcla de los bytes de las columnas.

Esta transformación considera las columnas de bytes como polinomios cuyos coeficientes pertenecen a GF(2^8), es decir, son también polinomios.

La función MixColumn consiste en multiplicar las columnas de bytes módulo x^4+I por el polinomio c(x).

Matemáticamente c(x) viene representado por: $c(x) = '03' x^3 + '01' x^2 + '01'x + '02'$

Este polinomio c(x) es primo con x^4+I, lo que permite que sea invertible. En forma algebraica esta función se puede representar como:

$$s'(x) = c(x) \otimes s(x)$$

Donde s'(x) representa la matriz de Estado resultante de esta transformación y s(x) la matriz de Estado entrante, producto de la transformación anterior.

Esta fórmula queda mejor expresada de forma matricial como se muestra en la, donde "c" representa el índice de la columna que se procesa:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figura 16 Representación matricial de función MixColumns [7]

Desarrollando la matriz, observamos claramente como cada byte nuevo de la matriz de Estado es una combinación de varios bytes de las distintas filas que forman una columna específica:

Para descifrar o invertir esta transformación, se deberá realizar el mismo procedimiento descrito pero con el polinomio d(x), que es el inverso de c(x).

La inversa de MixColumn se obtiene multiplicando cada columna de la matriz de estado por el polinomio d(x):

$$d(x) = \text{'0B'}x^3 + \text{“0D”}x^2 + \text{'09'}x + \text{'0E'}$$

Este polinomio, permite realizar la inversa del polinomio c(x), cumpliendo que:

$$c(x) \otimes d(x) = \text{'01'}$$

La transformación inversa se podría mostrar de forma matricial como se indica en la Figura 17:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figura 17 Representación matricial de InvMixColumns

2.2.3.4 Función AddRoundKey

En el paso AddRoundKey, la matriz de estado que vamos obteniendo de los pasos anteriores se combina con la matriz subclave mediante una operación or-exclusiva entre cada byte y el correspondiente de la subclave. Este paso se muestra esquemáticamente en la Figura 18.

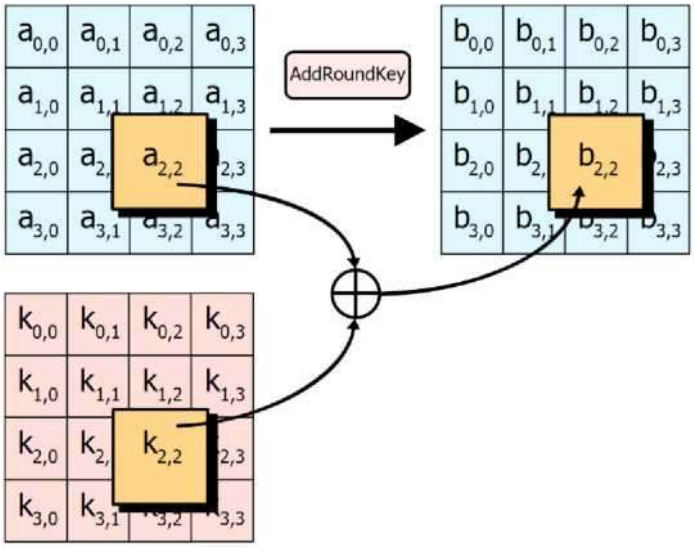

Figura 18 Función AddRoundKey [12]

El bloque resultante de esta transformación, será la nueva matriz de Estado para la siguiente ronda. Si la vuelta es la última será el bloque de salida,

Para aplicar esta transformación se debería coger una subclave para la vuelta actual, y formar una matriz con el mismo número de filas y columnas que la matriz de Estado con la que se está.

El bloque resultante de esta función será la nueva matriz de estado para la siguiente ronda.

La esencia de esta función recae sobre las subclaves. El algoritmo AES utiliza diferentes subclaves para que el resultado del algoritmo dependa completamente de una información externa al sistema: la clave de usuario. Así sigue el principio de la criptografía moderna, que establece que la seguridad de un algoritmo solo debe depender de la clave utilizada.

El funcionamiento de la transformación AddRoundKey es sencillo, lo verdaderamente interesante es conocer el procedimiento para generar las diferentes subclaves para cada ronda (*RoundKey*), subclaves que se derivan de la clave principal. Para ello se utiliza la función *KeyExpansion*.

La estructura del algoritmo Rijndael ya es conocida, consistiendo su funcionamiento en la reiteración de las funciones matemáticas que se definen en esta estructura. Son estas reiteraciones las que se conocen como "vueltas o rondas".

El número de vueltas se determinó, buscando el mínimo número de vueltas necesarias para ofrecer un margen de seguridad considerable frente a los ataques conocidos.

Por ejemplo, para un tamaño de bloque y de clave de 128 bits se utilizan 10 vueltas, esto es así porque no se han encontrado atajos en ataques para versiones reducidas con más de 6 vueltas. A estas 6, los autores, le suman otras 4 vueltas como margen de seguridad. Esto es un mecanismo de precaución, porque:

Dos vueltas de Rijndael producen una difusión completa, en el sentido de que, cada bit del estado depende de todos los bits de las 2 vueltas anteriores, es decir, un cambio en un bit del estado es similar a cambiar la mitad de los bits del estado después de dos vueltas. La alta difusión de una vuelta de Rijndael es gracias a su estructura uniforme que opera con todos los bits del estado. Para cifradores tipo Feistel (como por ejemplo. DES), un vuelta sólo opera con la mitad de los bits de estado y una difusión completa se obtiene en el mejor de los casos después de 3 vueltas y en la práctica 4 o más [7].

En general, ataques como el criptoanálisis lineal y diferencial aprovechan el rastro dejado a través de "n" vueltas para atacar "n+1" o "n+2" vueltas. Este es además el caso del ataque *Square* que usa 4 vueltas de la propagación del algoritmo para atacar 6 vueltas. En este sentido, se suman 4 vueltas para actualmente doblar el número de vueltas mediante las cuales se pueden encontrar pista, rastros que faciliten este tipo de ataques.

Para versiones de Rijndael con la clave más grande, el número de vueltas es incrementado por uno por cada 32 bits adicionales de la clave de cifrado. El principal objetivo es evitar *atajos* más efectivos que un ataque por fuerza bruta. La cantidad de trabajo para una búsqueda exhaustiva de la clave crece a medida que ésta crece también con lo que los *atajos* son menos eficientes para claves más grandes. Ataques con claves relacionadas permite el conocimiento de bits de la clave de cifrado o la habilidad para aplicar diferentes claves de cifrado. Si la clave de cifrado crece, las posibilidades disponibles para el criptoanalista disminuyen.

Para versiones con un tamaño de bloque mayor de 128 bits, el número de vueltas es incrementado por uno por cada 32 bits adicionales en el tamaño de bloque, ya

que a medida que crece el tamaño del bloque, disminuye la difusión completa del estado en una vuelta. El mayor tamaño del bloque produce rangos de posibles patrones que pueden ser aplicados a la entrada/salida de una secuencia de una vuelta para incrementarla. Esta flexibilidad añadida podría facilitar ataques.

2.2.3.5 Función *KeyExpansion*

La función "Expansión de clave" o *KeyExpansion*, permite derivar de la clave de cifrado, subclaves para cada vuelta. Su utilidad reside en permitir la resistencia contra los siguientes tipos de ataques:

- Ataques en los cuales parte de la clave de cifrado es conocida por los criptoanalistas.
- Ataques donde la clave de cifrado es conocida o puede ser elegida, por ejemplo, si el cifrador es usado como la función de comprensión de una función hash.
- Ataques por clave relacionada. Se deben evitar que distintas claves de cifrado puedan producir un gran conjunto de subclaves comunes para cada vuelta.

Además de esto, la función de "Expansión de clave" juega un papel vital en la eliminación de la simetría de la estructura del algoritmo, es decir:

- Simetría en una ronda: La ronda de transformación trata todos los bytes de un Estado en general de la misma forma. Esta aparente simetría es alterada gracias a las subclaves que se obtienen para cada vuelta.
- Simetría entre rondas: La ronda de transformación es idéntica para todas las vueltas o rondas. Esta igualdad es alterada obteniendo subclaves que dependen de cada ronda.

La eliminación de la simetría y la propia estructura del algoritmo permiten que resulte prácticamente imposible la existencia de claves débiles o semi-débiles como en otros

algoritmos criptográficos como DES, IDEA, etc. Por tanto, el algoritmo no tiene restricción en utilizar cualquier clave dentro del espacio de claves permitido.

Teniendo en cuenta todo esto, se puede resumir que la función *KeyExpansion* ha sido diseñada teniendo en cuenta los siguientes criterios:

- Usará una transformación invertible. Por ejemplo, el conocimiento de cualquiera Nk palabras consecutivas de la clave expandida permitirá regenerar todo la matriz de bytes de subclaves.
- Rapidez en un gran abanico de procesadores.
- Proporcionar constantes dependientes de cada ronda que elimine la simetría.
- El conocimiento de parte de la clave de cifrado o bits de una subclave de una ronda no permitirá calcular muchos bits de otras subclaves.
- Proporcionar no-linealidad que prohíba determinar diferencias en la subclave de cada ronda a partir de diferencias de la clave de cifrado.
- Sencillez de diseño.

La función de expansión de clave permite generar bytes útiles como subclaves a partir de la clave de sistema K. Esta función de expansión se puede describir como una matriz lineal, denominado W, de palabras de 4x4

Las primeras N_k palabras de esta matriz contienen la clave de cifrado, ya que la clave del usuario se mapea tal cual a la matriz W, mientras que el resto de palabras se van generando a partir de estas primeras N_k palabras como se indica en la Figura 19.

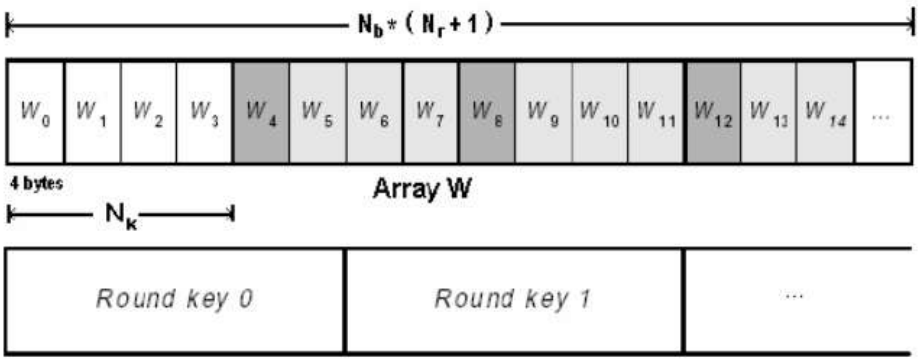

Figura 19 Función Expand Round Key [12]

Se observa cómo la función de expansión de clave depende del valor de N_b. Ante este hecho los autores definieron dos versiones para esta función, una para $N_b < 6$ y otra para $N_b > 6$, por motivos de seguridad. Estas versiones se describen el código mostrado en la Figura 20, utilizando el lenguaje de programación C.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Figura 20 Código Expand Round Key

En el código mostrado en la Figura 20 se observa cómo se mapea directamente la clave del usuario a la matriz W. Se mapean Nk palabras. El resto del funcionamiento permite generar bytes para subclaves. En este proceso entra en juego la función *SubWord* que devuelve el resultado de aplicar la S-BOX de Rijndael a cada uno de los bytes de los 32 bits de la palabra que se le pasa como parámetro, la función *RotWord* que rota una posición a la izquierda los bytes de la palabra, de tal forma que si se le pasa como parámetros la palabra de 4 bytes (a,b,c,d) devuelve (b,c,d,a), y la función *Rcon* que genera una constante teniendo en cuenta que:

$$Rcon(j)=(R(j),0,0,0)$$

Cada R(j) es el elemento $GF(2^8)$ correspondiente al valor x^{j-1}

Siguiendo todos los criterios descritos en esta función se podrían generar todos los bytes de subclaves necesarios para la versión del algoritmo con la que se trabajará.

2.2.4 Descripción del algoritmo AES para Descifrado

El proceso de descifrado del algoritmo Rijndael consiste en sustituir las transformaciones utilizadas en el cifrado por sus inversas, tal y como se han descrito en la sección anterior, e invertir el orden de aplicación de dichas transformaciones o funciones matemáticas.

Teniendo en cuenta esto la estructura del algoritmo de descifrado sería:

- Vuelta Final:
    - *InvAddRoundKey*
    - *InvShiftRow*
    - *InvByteSub*

- Vuelta Estándar:
    - *InvAddRoundKey*
    - *InvMixColumn*;
    - *InvShiftRow*(Estado);
    - *InvByteSub*(Estado);
- Salida:
    - InvAddRoundKey(Estado,RoundKey);

De esta forma tan sencilla se construye el algoritmo para descifrar una información cifrada con el algoritmo Rijndael

2.2.5 Modos de ejecución AES

El funcionamiento de un algoritmo de cifrado por bloques comentado hasta ahora, solo permite cifrar un mensaje del tamaño de un bloque de cifrado. Si se desea cifrar algo con mayor longitud se debe recurrir a los modos de operación.

Los modos de operación son una herramienta que permite usar repetidamente un algoritmo de cifrado con una única clave para cifrar de manera segura mensajes de longitud variable.

Antes de cifrar un mensaje se debe dividir los datos en distintos bloques de cifrado. Posteriormente, se deberán seguir las directrices del modo de funcionamiento para cifrar todos los bloques por separado.

Cada modo de funcionamiento tiene diferentes directrices que aportan mayor o menor seguridad, eficacia o eficiencia.

Los modos principales de funcionamiento que se pueden usar, y que pasamos a describir a continuación, son: *ECB, CBC, OFB, CFB y CTR*.

2.2.5.1 ECB (*Electronic codebook*)

Es el modo de funcionamiento elegido en este proyecto. Divide el mensaje en los distintos bloques y codifica por separado a todos con la misma clave. Su diagrama de flujo se muestra en la Figura 21.

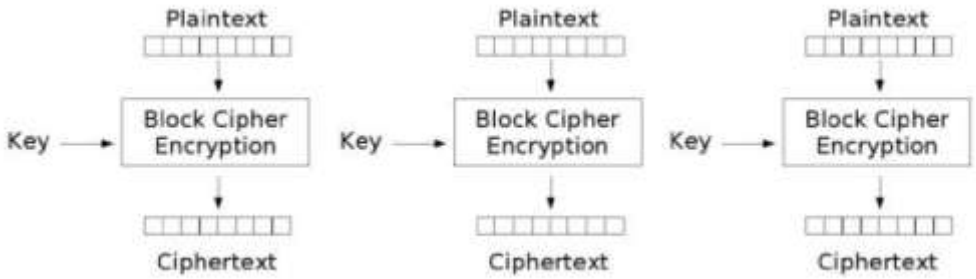

Figura 21 Modo de funcionamiento ECB [14]

La debilidad de este modo es que codifica bloques iguales de la misma manera, por lo que no esconde correctamente los patrones de los datos.

2.2.5.2 CBC (*Cypher-block chaining*)

En el modo CBC antes de cifrar cada uno de los bloques, se les aplica una operación XOR con el bloque cifrado anterior.

Además, si se pretende que este mensaje cifrado sea único, se le puede aplicar una operación XOR al primer bloque junto con un vector de iniciación que debería conocer quién vaya a decodificar el mensaje para recuperar el primer bloque.

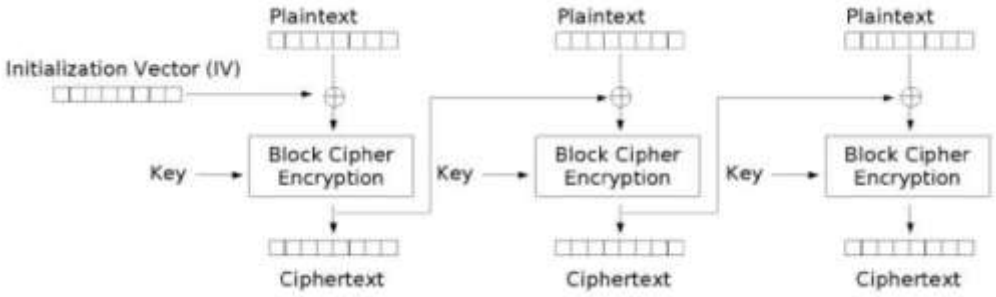

Figura 22 Modo de funcionamiento CBC [14]

2.2.5.3 OFB (*Output feedback*)

El modo de funcionamiento OFB convierte el cifrado en bloque en un cifrado de flujo. En él no se codifica directamente con AES el mensaje, si no que se parte de un vector de inicialización que es codificado mediante AES y el vector cifrado se suma al primer bloque a codificar mediante una operación XOR. El resto de bloques se codifican de la misma manera solo que cifrando en bloque la salida cifrada del anterior codificador de bloque en lugar del vector de inicialización como se muestra en la Figura 23.

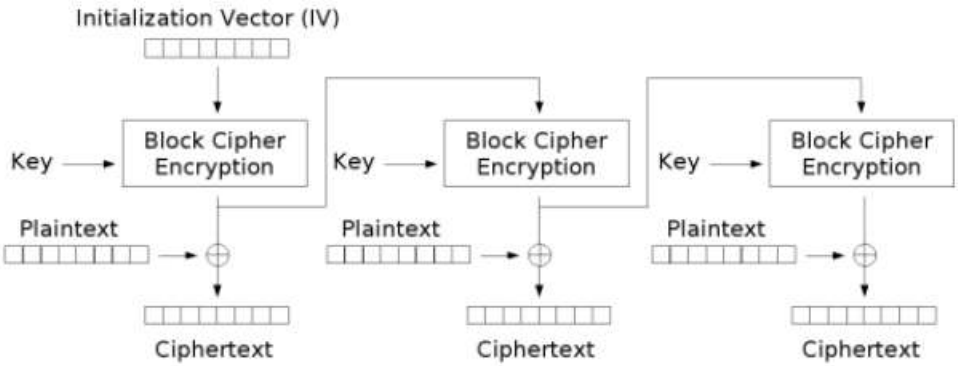

Figura 23 Modo de funcionamiento OFB [14]

2.2.5.4 CTR (*Counter*)

En el modo CTR se parte de una secuencia o un contador cuyos valores son codificados mediante el algoritmo de bloque. Estos valores codificados se suman a los bloques de texto a cifrar mediante una operación or-exclusiva (Figura 24).

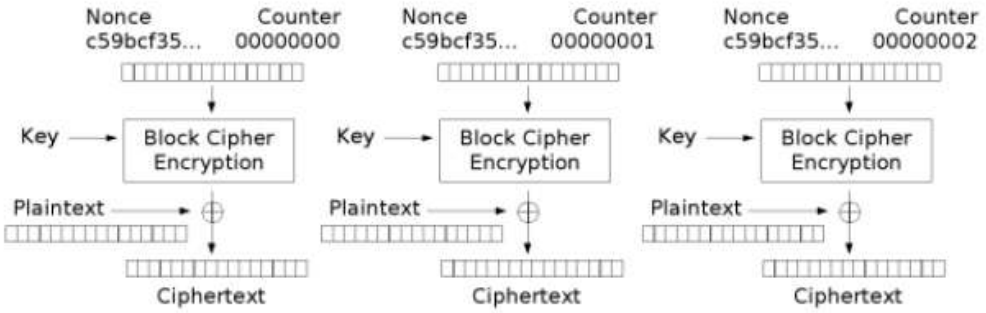

Figura 24 Modo de funcionamiento CTR [14]

3.Entorno de Trabajo

En este capítulo se describe el entorno de trabajo utilizado en el presente proyecto. Se comienza con una descripción de la placa STM32l-Discovery y de sus diferentes componentes para luego describir las aplicaciones software utilizadas para el desarrollo y la depuración del código, así como una descripción de los demás dispositivos hardware que se han utilizado

3.1 Placa STM32L-DISCOVERY

El kit STM32L-DISCOVERY, mostrado en la Figura 25, es una tarjeta de bajo coste y consumo, con capacidad para desarrollar, evaluar y crear proyectos con un sistema microcontrolador de la familia STM32. Está basado en un microprocesador STM32L152RBT6, con una arquitectura *ARM Cortex–M3* e incluye un programador/debugger ST-Link/V2, un *Display LCD* de 24 segmentos, *LEDs*, pulsadores y un sensor táctil lineal [15].

Figura 25 Placa STM32L-Discovery [16]

En las siguientes secciones se pasa a describir de forma general las características y componentes de esta placa de evaluación.

3.1.1 Características generales y componentes

A continuación se indican de forma esquemática las principales características de la placa Discovery. Aquellos componentes que se utilicen en el entorno de trabajo desarrollado serán explicados con mayor detenimiento:

- El corazón del dispositivo está basado en un Microcontrolador STM32L152RBT6, con un procesador ARM Cortex–M3 de 32 bits, 128 KB de memoria Flash, 16 KB de RAM, y memoria EPROM de 4KB. Todo ello en un chip de 64 pines.
- Dispone de un programador y debugger interno, ST-Link/V2, con capacidad de selección del modo de funcionamiento.
- La alimentación de la placa puede ser a través de un conector USB o con una fuente de alimentación externa de 3.3 o 5 V.
- Para mostrar medidas y otros parámetros, lleva incorporado un LCD DIP28, de 24 segmentos y 4 comunes.
- Cuatro LEDs.
- Dos botones (Reset y botón de usuario).
- Un sensor linear y cuatro teclas para selección de modos.
- Medidor de Corriente I_{DD} para conocer el consumo.

Para un mejor entendimiento del funcionamiento de la placa, se expone en la Figura 26 un diagrama de bloques del hardware de la misma:

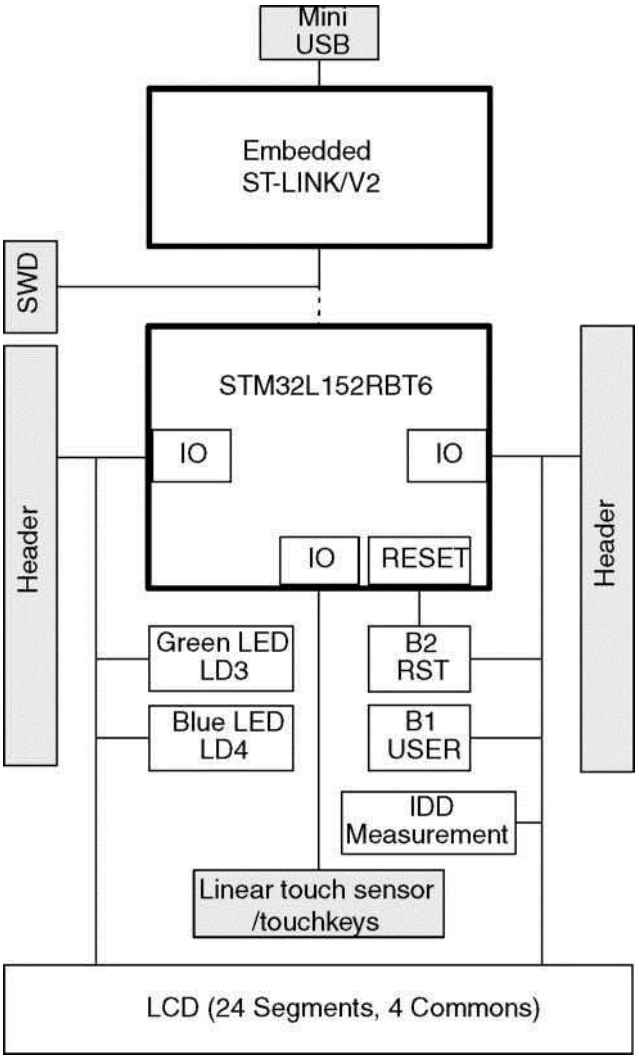

Figura 26 Diagrama de elementos hardware [16]

La familia STM32L152xxx incorpora la capacidad de conectividad por USB (*Universal Serial Bus*), con el procesador *ARM Cortex* que opera a 32 MHz de frecuencia, una unidad de protección de memoria (MPU), memorias Flash, RAM de gran velocidad y una gran cantidad de periféricos conectados a dos buses APB. Disponen de un conversor analógico digital (ADC) de 12 bits, además de timers y comparadores, etc.

Esta familia contiene, además, interfaces de comunicación estándar y avanzados, tales como dos *I^2C* y *SPI*, tres *USART* y un *USB*, así como un reloj en tiempo real y un conjunto de registros de backup que mantienen su valor en el modo *Standby*.

A continuación describiremos con más destalle cada uno de los componentes principales del sistema.

3.1.2 Programador/*debugger*

En la placa STM32L-DISCOVERY está integrada la herramienta ST-Link/V2. Esta pieza realiza las funciones de programador y *debugger* (depurador), pudiéndose utilizar para dos funciones diferentes, dependiendo de la posición de los *Jumpers* de la placa:

- Programar y depurar el micro de la placa.
- Programar y depurar el micro de una aplicación externa, utilizando un cable conectado a CN2.

El ST-Link/V2 solamente funciona con el interface *Serial Wire Debug* (SWD). Esta pieza supone un reemplazo para el *JTAG* en situaciones en las que el número de pins disponibles es limitado, generalmente debido a los encapsulados de reducido tamaño. El SWD permite todas las opciones de Debug y Test del JTAG reduciendo los pines necesarios de 5 a 2. Permite un acceso a la memoria de sistema en tiempo real sin requerir recursos propios del micro.

3.1.3 Fuente de alimentación

Para esta placa, la alimentación se suministra bien desde el ordenador, vía cable USB, o mediante una fuente exterior de 3V o 3,3 V.

3.1.4 Display de Cristal Líquido LCD (*Liquid Crystal Display*)

Esta placa dispone de un display LCD representado en la Figura 27, de seis dígitos de 14 segmentos y cuatro barras para mostrar cualquier información, utilizando todos los COM´s. Este dispositivo también funciona con seis dígitos de 8 segmentos utilizando solo COM0 y COM1. Esta última configuración permitiría usar COM2 y COM3 como puertos de entrada/salida.

Características principales:

- 14 segmentos y 4 COM´s
- *Método de control*: multiplexado 1/4 *duty*, 1/3 *bias* (4 pines para COMs y 1 pin cada 4 segmentos)
- Voltaje de operación: 3 V
- Rango de temperatura de operación: 0 a 50°C
- Conector: 28-pin DIL 2.54 mm

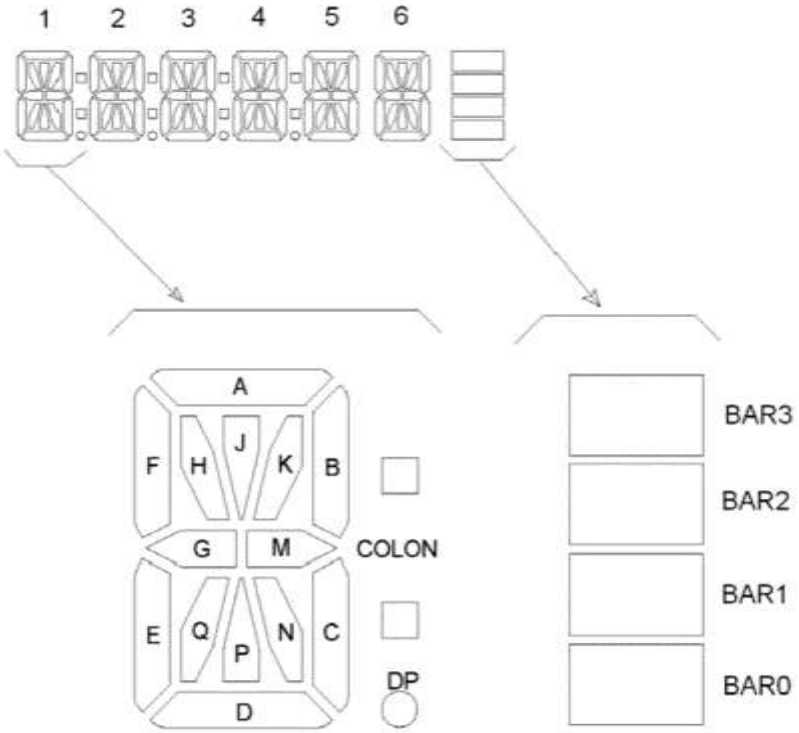

Figura 27 Distribución de los segmentos del LCD

3.1.5 LEDs

La placa STM32L-Discovery está equipada con cuatro LEDs:

- LD1 COM: Su estado por defecto es rojo. Conmuta a verde para indicar que se están realizando comunicaciones entre el PC y el *ST-LINK/V2*.

- LD2 PWR: Si está iluminado en rojo, indica que la placa está alimentada.
- LD3 GREEN: Se ilumina en verde. Es un LED controlado por software para uso del usuario. Está conectado a la I/O PB7.
- LD4 BLUE: Se ilumina en azul. Es un LED controlado por software para uso del usuario. Está conectado a la I/O PB7.

3.1.6 Botones

La placa cuenta con 3 botones/pulsadores:

- B1 USUARIO. Botón de usuario conectado al I/O PA0 de la placa STM32L152RBT6
- B2 RESET. Botón de usuario utilizado para resetear la placa
- La placa STM32L-DISCOVERY incluye un sensor táctil lineal que puede ser usado como un sensor táctil lineal de tres posiciones o como cuatro teclas.

3.2 Microcontrolador STM32L152RBT6

La placa STM32L-DISCOVERY está basada en el microcontrolador STM32L152RBT6 de 32 bits que pertenece a la familia STM32L, la cual presenta una orientación clara al logro de un reducido consumo, perteneciendo a la plataforma *EnergyLite*. Combina este reducido consumo con una gran eficiencia dando lugar a un gran ahorro energético.

En tecnologías CMOS como ésta, parte de la energía se emplea en la carga y descarga de pequeñas capacidades intrínsecas a dicha tecnología. Los microcontroladores de la familia STM32 han sido diseñados con la tecnología propietaria de *130 nm*, que al ser de dimensiones menores, reduce las capacidades de los nodos reduciendo por tanto la energía consumida.

En la Figura 28 se muestra el diagrama de bloques del microcontrolador donde se exponen todos los componentes internos y periféricos que contiene:

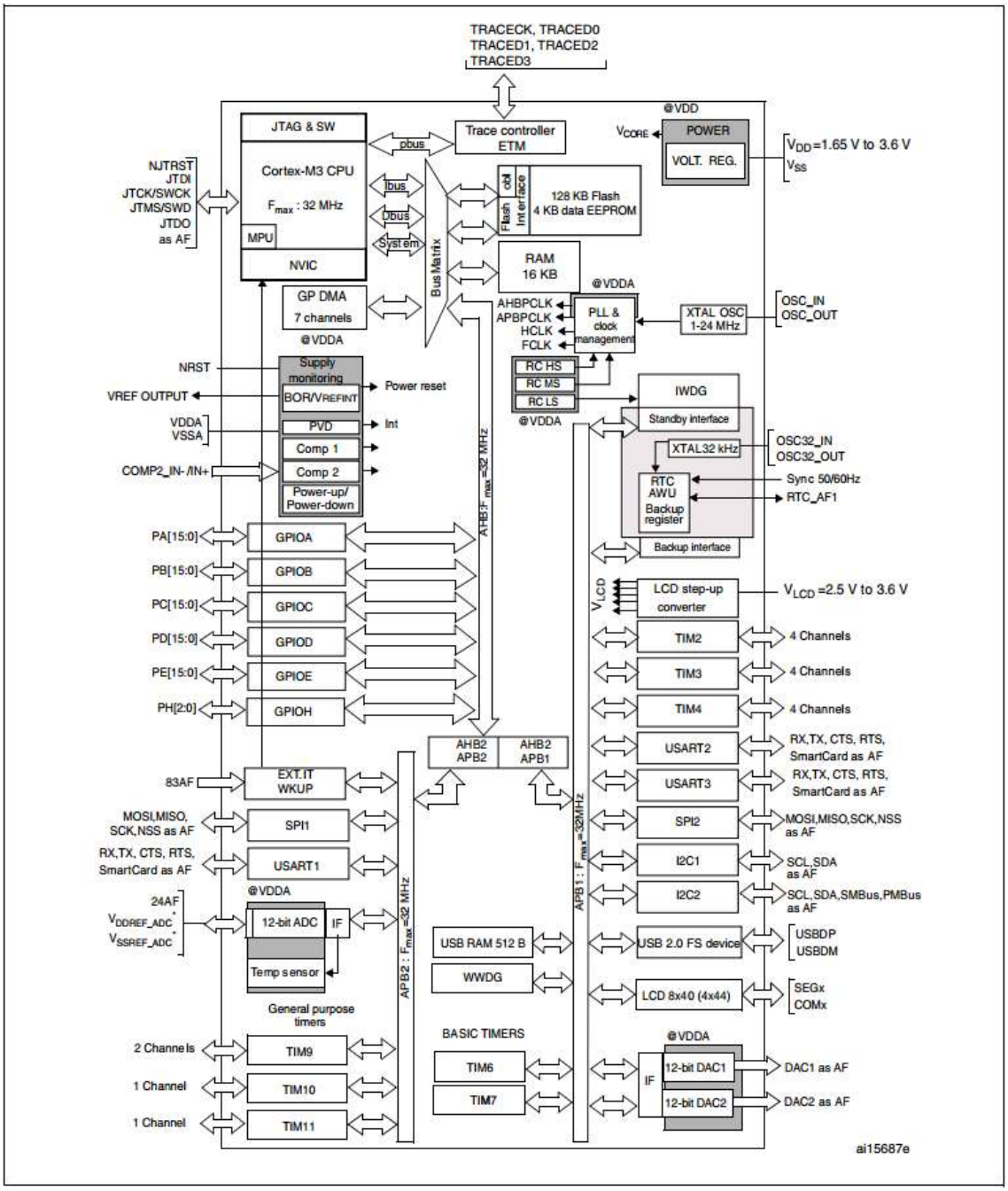

Figura 28 Diagrama de bloques de micro STM32L152RBT6 [16]

Las principales características del microcontrolador STM32L152RBT6 se podrían resumir en:

- Diseño para consumo muy bajo de potencia. Consumo de potencia nulo en periféricos que no se usan. Dispone de hasta seis modos de bajo consumo disponibles.

- Sistema de reloj avanzado y flexible (múltiples fuentes de reloj internas y externas). Los relojes pueden ser conmutados y ajustados en frecuencia dependiendo de las necesidades.
- Accesos de memoria directos en la placa (hasta 12 canales DMA)
- Fuente de alimentación de 1.65 V a 3.6 V
- Rango de funcionamiento de temperatura de -40°C a 85°C/105°C
- Corriente de funcionamiento reducida en los diferentes modos de funcionamiento:
    - Modo *Standby* + RTC: 0.9 µA
    - Modo *Stop*: 0.57 µA
    - Modo *Standby* + Stop: 0.2 µA
    - Modo *Run* bajo consumo de 9 µA
    - Modo *Run* 214 µA/MHz
- Tiempo de respuesta < 8 µs
- Detector de voltaje programable (PVD)
- Oscilador de cristal de 24 MHz
- PLL para reloj de la CPU y USB (48 MHz)
- USART
- Diseñado para desarrollo
- Diseñado para debug
- Hasta 83 I/Os rápidas (73 I/Os toleran 5V), todas mapeables en 16 vectores de interrupción externos
- Registro de Backup de 80 Bytes
- Driver de LCD de hasta 8x40 segmentos, con:
    - Ajuste de contraste
    - Modo parpadeo
- Conversor de *setup* en la placa
- 2 comparadores de ultra bajo consumo
- Controlador de DMA de 7 canales
- Interface de comunicación para 8 periféricos

3.2.1 ARM Cortex-M3

El micro STM32L152RBT6 cuenta con una arquitectura ARM Cortex-M3. Esta arquitectura presenta un gran rendimiento computacional mientras que alcanza una optimización y reducción de la energía consumida, tanto dinámica como estática. Esta arquitectura de 32 bits presenta una mayor densidad de código que sus predecesoras de 8 y 16 bits, permitiendo una reducción en los requerimientos de memoria y maximizando el uso de la memoria Flash integrada [17].

Dentro de las ventajas de la arquitectura Cortex-M3 cabe destacar 3 elementos principales, la capacidad de gestión de interrupciones a través del NVIC (*Nested Vectored Interrupt Controller*), la MPU (Unidad de Protección de Memoria) que mejora la fiabilidad definiendo los atributos de memoria por diferentes regiones de la misma, y el *CMSIS* que permite una capa de abstracción adicional al usuario. Las funciones y ficheros del CMSIS serán explicados con más detalle en el siguiente capítulo.

Además de estas 3 características principales, podemos destacar de la arquitectura ARM Cortex M3 los siguientes aspectos:

- Su alto Rendimiento
- Los buses de datos y de instrucción son independientes, lo que permite realizar accesos simultáneos a código y datos.
- Set de instrucciones Thumb-2. No es necesario cambiar entre estados ARM (32 Bits) y Thumb (16 bits), por lo que se reducen los ciclos de instrucción y el tamaño del programa. Muchas instrucciones, como la multiplicación son de un solo ciclo.
- Gracias al juego de instrucciones Thumb-2, Cortex-M3 ofrece mayor densidad de código y reduce los requerimientos de memoria.
- Cortex-M3 implementa relojes que pueden opera a más de 100 Mhz. Además, tiene una relación reloj por instrucción (CPI) mejor que el resto de procesadores.
- Funciones Avanzadas en el Manejo de Interrupciones gracias al NVIC:
  - Gestión de 240 interrupciones.

- El estado del procesador se guarda automáticamente.
- El NVIC puede programar la prioridad de interrupciones individualmente. Maneja hasta 255 niveles de prioridad.
- Dispone de una NMI (*NonMaskeable Interrupt*) que garantiza la ejecución del manejador asociado a esta interrupción. Esto es importante en aplicaciones de seguridad críticas.

- Bajo Consumo de Energía
  - Es óptimo para diseños de bajo consumo
  - Soporta diversos modos de ahorro de energía.

3.2.2 Funcionamiento del reloj

El controlador de reloj distribuye las señales de reloj, producidas en los diferentes osciladores, al micro y a los periféricos.

Las principales características del reloj son:

- Dispone de pre-escalado programable para el reloj lo que permite obtener una mejor relación velocidad-consumo.
- Las fuentes de reloj pueden ser conmutadas en tiempo real en el modo Run a través de un registro de configuración.
- Maneja tres fuentes de reloj para sincronizar el reloj master.
- Tiene un circuito PLL, que permite, entre otras aplicaciones, dar una salida de reloj de 48 MHz usada para el interfaz USB.

El reloj en tiempo real (RTC), es un reloj/contador BCD independiente. Existen unos registros dedicados que contienen el segundo, minuto, hora (12/24), así como los días del mes. Realiza una gestión automática de los días del mes: 28, 29, 30 y 31 dependiendo del mes y del año si es o no bisiesto [18].

3.2.3 GPIOs

Este periférico consiste en los puertos de entrada salida denominados de propósito general ya que cada pin se puede programar por software para ser una salida, una entrada o una función alternativa de un periférico. La mayor parte de estos pines están repartidos entre funciones alternativas analógicas o digitales y pueden ser re-mapeados individualmente.

En el siguiente capítulo se entrará en más detalle sobre la configuración y uso de los diferentes pines de la placa.

Existe un controlador de eventos/interrupciones externo (EXTI) que se basa en 23 líneas de detección de flancos para generar las peticiones de eventos/interrupciones. Cada línea puede ser configurada individualmente.

3.2.4 Memorias

El microcontrolador dispone de una memoria RAM de 16 Kbyte accesible a la velocidad de reloj de la CPU.

Por otra parte, la memoria no volátil se divide en tres configuraciones:

- 32, 64 o 128 Kbyte de memoria Flash
- 4 Kbyte de EPROM de datos
- Bytes opcionales

Este sistema dispone de un controlador de propósito general DMA (*Direct Memory Access*) de 7 canales, capaz de gobernar todas las transferencias memoria-a-memoria, periférico-a-memoria y memoria-a-periférico. El DMA dispone de un controlador de *buffer* circular que evita interrupciones cuando se alcanza el final del *buffer*.

Este controlador de DMA se puede usar con los principales periféricos: SPI, I^2C, USART, relojes de propósito general y ADC.

En la Figura 29 se muestra otro esquema de los componentes del micro, más simplificado, donde se puede apreciar mejor la distribución la memoria y los buses.

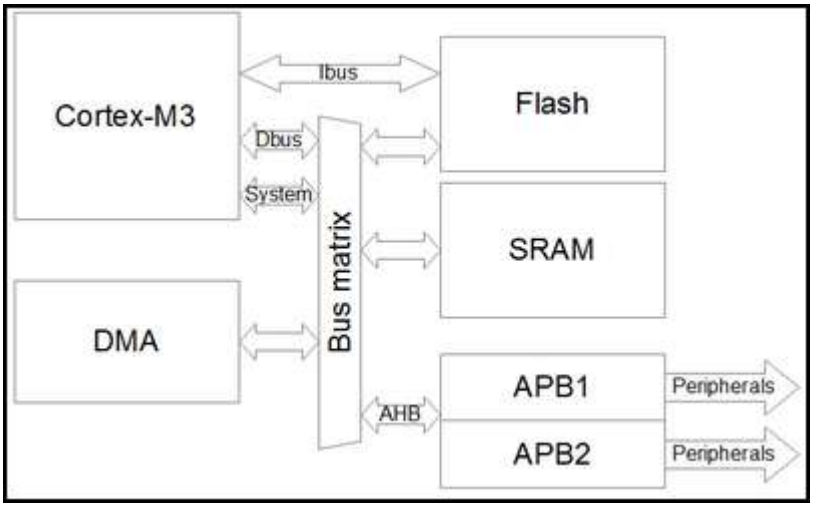

Figura 29 Diagrama de bloques simplificado del micro STM32L152RBT6 [19]

3.2.5 Conversor Digital Analógico (DAC)

Dos canales de DAC están disponibles para convertir señales digitales en señales de salida analógicas, esto es, niveles de voltaje. Se ha diseñado con una estructura compuesta por una cadena de resistencias integradas y un amplificador en configuración no invertida.

Este conjunto de conversores disfruta de las siguientes características:

- Dos convertidores DAC, uno para cada canal de salida.
- Alineación de los datos por la derecha o izquierda, en modo 12 bits.
- Capacidad de sincronización.
- Generación de onda de ruido.
- Generación de onda triangula.
- Conversiones independientes o conversión dual.
- Movimiento de datos con DMA.
- Disparos externos para conversión.
- Entrada de voltaje de referencia V_{REF+}

3.2.6 Comparadores

Esta placa dispone de dos comparadores que comparten la misma corriente en vacío y el voltaje de referencia. Este voltaje puede ser interno o externo (a través de un pin I/O).

Uno de estos comparadores tiene un umbral fijo y el otro puede ser uno de los listados a continuación:

- Salida DAC.
- I/O Externo.
- Voltaje de referencia interno (Vrefint) o un submúltiplo.

3.2.7 Interfaces de comunicación

El microcontrolador STM32L152RBT6 cuenta, gracias a las ventajas de la arquitectura Cortex-M3 con diversos medios para establecer comunicaciones con otros dispositivos. A continuación se presentan los cuatro componentes que permiten estas comunicaciones.

- BUS I2C: Pueden operar hasta dos buses I2C, en modo master o esclavo. Los movimientos de memoria pueden ser gestionados por el DMA y soportan SM Bus 2.0/PM Bus.
- USART (*Universal Synchronous/Asynchronous Receiver Transmiter*): Todos los interfaces USART pueden comunicarse a velocidades de hasta 4 Mbit/s. Todas las USART pueden ser usadas con el controlador de DMA.
- SPI (*Serial Peripheral Interface*): Se dispone de dos SPI´s para comunicarse hasta a 16 Mbits/s en configuración master o esclavo, en los modos de comunicación *full-duplex* y *half-duplex.* El DMA es capaz de realizar los movimientos de memoria para ambos SPIs.
- USB (*Universal Serial Bus*): El micro STM32L152RBT6 dispone de un USB compatible con el USB de 12 Mbits/s. El interface USB implementa el interface de máxima velocidad (12 Mbits/s)

3.2.8 Conversor Analógico Digital (ADC)

Como su nombre indica es el encargado de transformar una señal analógica (nivel de tensión) en un valor digital equivalente.

El micro STM32L152RBT6 dispone de solo 1 ADC con 26 canales (2 de ellos internos para medir corriente y temperatura). Cada canal es capaz de recibir una señal distinta y transformarla. Este ADC cuenta con una resolución de 12 bits, es decir, los valores transformados estarán comprendidos entre 0 y 4096.

El ADC trabaja con la frecuencia del HSI (*High Speed Internal Oscillator*) de 16MHz, lo que le da independencia respecto del micro, realizando las conversiones a máxima velocidad. Esto puede resultar en problemas si el resto del sistema no es capaz de procesar las medidas a tiempo, produciéndose una pérdida de información por desbordamiento. Para evitar esto se puede hacer uso tanto del pre-escalado del reloj (divide la frecuencia de entrada entre 2 o 4) o bien introduciendo retardos entre las conversiones.

El ADC puede trabajar en varios modos de funcionamiento:

- Modo Simple: El ADC realiza solamente una conversión y se detiene activando un *flag* de fin de conversión (*End of Conversion*) y generando, si así se ha configurado, una interrupción que indica al sistema que la medida está disponible para su consulta.
- Modo Continuo: El ADC comienza una nueva conversión tan pronto como ha terminado la anterior. Al igual que en el modo simple, el *flag fin de conversión* se pone a 1 y se genera una interrupción si anteriormente se habilitó.
- Modo Scan: El ADC realiza la conversión de un grupo de canales. Tras realizar la conversión de un canal pasa al siguiente y así sucesivamente hasta convertir todo el grupo de canales. Dependiendo de la configuración puede para en ese momento o volver a comenzar por el primer canal.

- Modo Discontinuo: Se establece un parámetro n (n<18) cuyo valor será el número de conversiones que realice el ADC por cada evento *trigger*.

Para poder interpretar el resultado, es necesario establecer la relación de conversión, que depende de la resolución del ADC y del voltaje máximo a medir:

$$Relación_{Conv} = \frac{\text{Señal Analogica máxima}}{\text{resolución ADC}}$$

3.3 Ordenador Personal

Para el desarrollo de este proyecto se hecho uso de un ordenador personal de 32bits trabajando con un sistema operativo *Windows 7.* El ordenador se ha elegido para satisfacer una serie de requerimientos mínimos para poder utilizar las diferentes herramientas de software que han sido necesarias para crear este entorno de trabajo. En las secciones siguientes se describen estas herramientas

3.3.1 Atollic TrueStudio for ARM Lite

Se trata de un *Entorno Integrado de Desarrollo* (IDE) basado en *Eclypse*, diseñado específicamente para microcontroladores ARM. Esta herramienta permite realizar una gestión completa de los proyectos gracias a sus diferentes componentes entre los que destacan un compilador de C/C++, un potente editor y una herramienta de depuración de código de carácter profesional así como herramientas de programación Flash para pasar la aplicación a la ROM. En la Figura 30 y Figura 31 se muestras las ventanas principales de edición de código y de depuración del mismo, respectivamente.

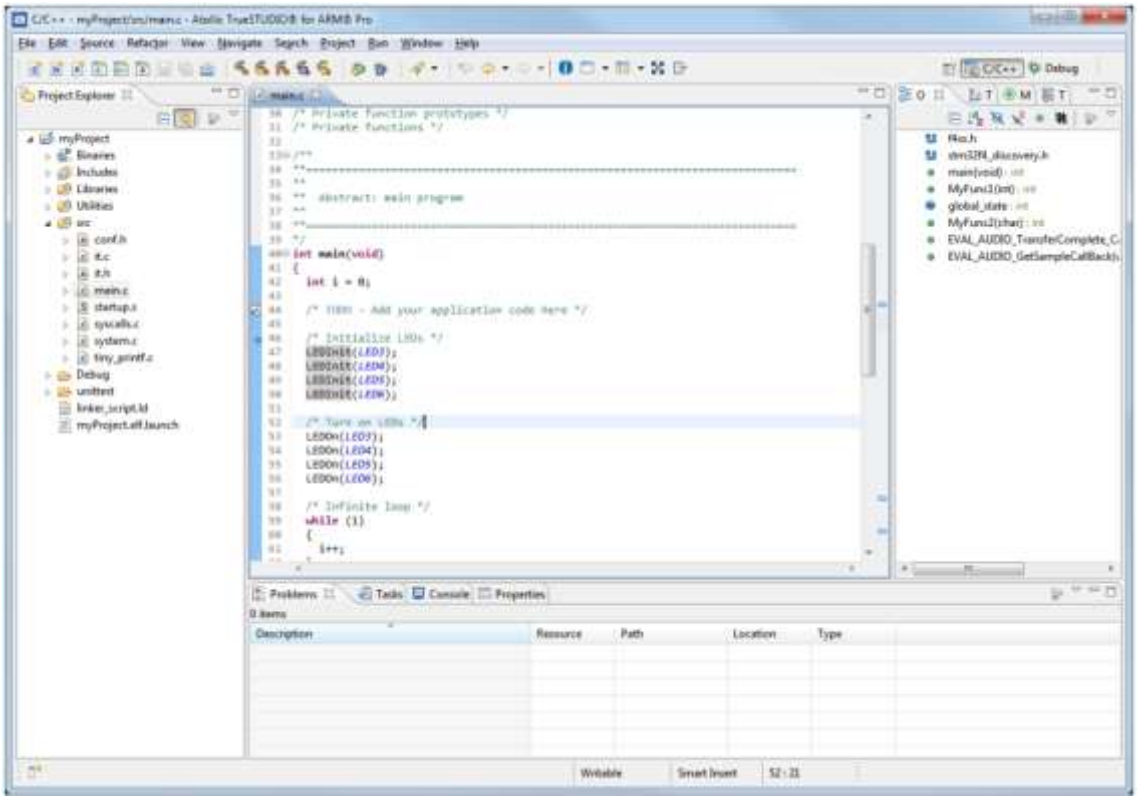

Figura 30 Atollic True Studio - Ventana de Edición

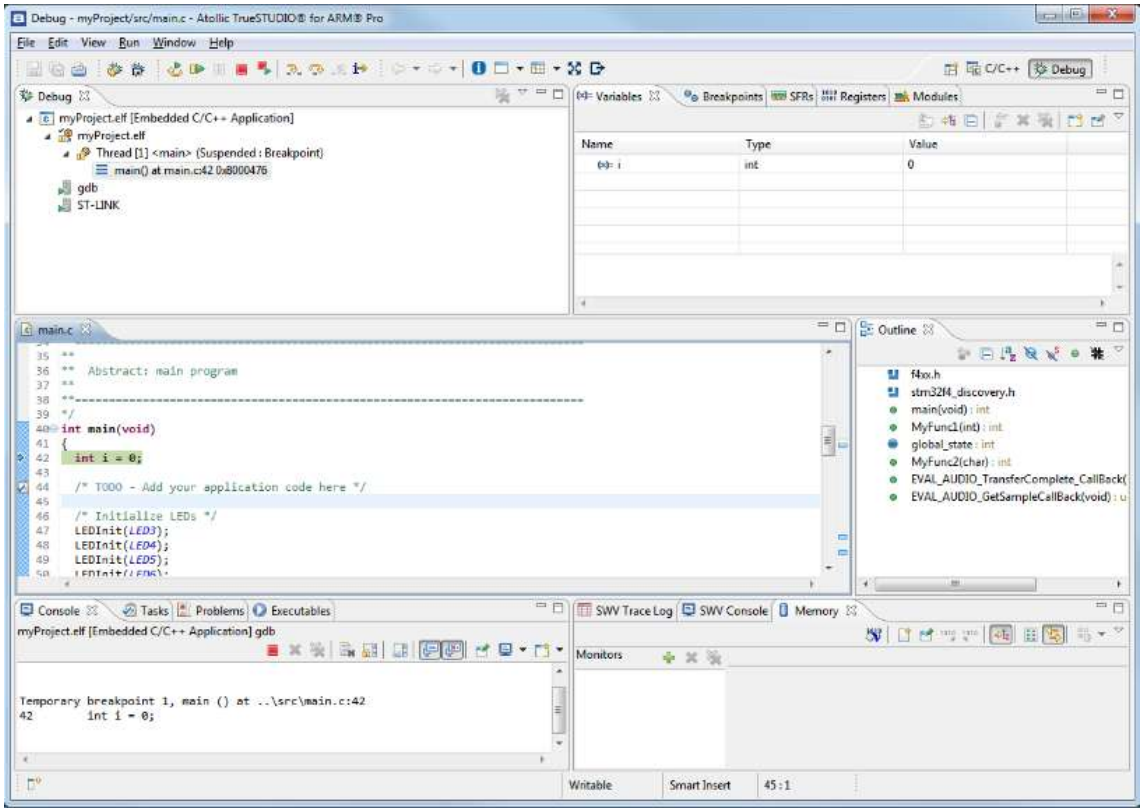

Figura 31 Atollic TrueStudio - Ventana de Depuración

Una de las razones principales por las que se ha seleccionado este IDE para el desarrollo es el hecho de que el fabricante ofrece un gran soporte a los microcontroladores STM y en concreto a la familia STM32. Este soporte consiste principalmente en la capacidad para autogenerar proyectos específicos para esta familia de micros incluyendo ya las configuraciones necesarias para empezar a desarrollar las aplicaciones. También cabe destacar una amplia gama de ejemplos de aplicaciones descargables desde la web con el propio IDE y soporte para *ST-link v2.*

Otro de los aspectos determinantes para su elección es que la versión de prueba, *Lite*, es gratuita y no tiene limitaciones en aspectos como el tamaño de código ni un límite temporal de prueba a diferencia de otros entornos de desarrollo. La diferencia respecto a la versión de pago es que al inicio se muestra un mensaje del fabricante que es necesario cerrar y también ciertas limitaciones en la depuración, aunque lo único destacable es el límite en el número de *Breakpoints* que se pueden incluir en el código para su depuración.

3.3.2 MATLAB

Se ha elegido Matlab como aplicación que orquestra la aplicación desde el PC haciendo de interfaz del usuario y presentando y almacenando la información recibida desde la placa.

Matlab permite realizar la comunicación vía puerto serie con la placa de una forma sencilla gracias a su amplia gama de funciones para dicho fin. Esto, sumado a su potencia en el tratamiento de datos, hace de Matlab una herramienta adecuada para las necesidades de este proyecto.

Cabe destacar su editor de scripts y funciones que incorpora un depurador de código que permite un desarrollo más ágil de los programas.

En el siguiente capítulo se expondrá más en profundidad el desarrollo realizado en esta herramienta y se expondrán algunas de sus bondades.

3.3.3 Otras aplicaciones

En esta sección se describen las aplicaciones que han servido de apoyo en la realización del proyecto aunque han desempeñado un rol más secundario, principalmente como ayuda a la depuración y detección de errores.

Hyperterminal

Es un terminal que viene incluido en el sistema operativo Windows y permite establecer una comunicación con el microcontrolador a través del puerto serie. Su ventaja principal es su fácil uso.

Esta aplicación se ha usado con fines exclusivos de depuración dada su sencillez en el establecimiento de la comunicación por puerto serie.

Rijndael Animation y Rijndael Inspector

Se trata de dos aplicaciones desarrolladas en flash que permiten entender de forma muy visual el funcionamiento del algoritmo AES. En especial el Rijndael Inspector ha sido de mucha utilidad a la hora de depurar el código AES ya que permite, para una matriz de entrada y una clave, ver todas las matrices de estado de todas las rondas así como las subclaves [11].

3.4 Cable Comunicación Serie TTL-232R 3V3WE

Este dispositivo realiza una conversión entre niveles de señal TTL y las señales del protocolo USB. Contiene un pequeño circuito electrónico basado en el FT232R que se encuentra integrado en el conector USB [20].

Este cable/conversor cuenta con drivers descargables y gratuitos desde la página web del fabricante, *FTDIChip,* compatibles con numerosos sistemas operativos y dispositivos.

Una vez instalados los drivers, estos hacen que al conectar el cable, sea reconocido como un puerto COM virtual (VCP) lo que permite al usuario comunicarse a través de dicho puerto.

En la Figura 32 se muestra una imagen de las 2 terminaciones del cable donde se ve el conector USB con el circuito impreso integrado y el otro extremo con los cables sin conector ya que se ha usado la versión WE (*wire ending*)

Figura 32 Cable TTL-232R 3V3WE [20]

4. Descripción de la aplicación

El desarrollo de la aplicación de medidas de consumo durante la ejecución de Cifrado/Descifrado usando el *Advance Encryption Standard* (AES) se ha realizado en dos partes diferentes. Por un lado se ha usado la placa anteriormente descrita STM32L-Discovery, como plataforma donde se ejecuta el código y se toman las medidas y por otro lado un desarrollo usando la herramienta MATLAB en el PC para procesar las medidas tomadas y actuar como interfaz con el usuario. En esta sección se describen en profundidad tanto el desarrollo en la placa como el desarrollo en MATLAB para posteriormente analizar el sistema de toma de medidas y un análisis de las mismas.

4.1 Desarrollo en la placa STM32L-Discovery:

En este apartado se describe como se ha realizado la configuración de la placa y sus diferentes componentes así como el desarrollo del algoritmo AES en lenguaje C tanto procesándose en 8 bits como en 32 bits.

4.1.1 Configuración de la placa:

La placa STM32L-Discovery incluye el micro STM32L152RBT6 que, como ya se explicó en un capitulo anterior, consta de diversos periféricos, algunos de los cuales se han usado para la configuración de este entorno de trabajo. En esta sección se describe el rol de cada uno de ellos en el contexto del sistema completo y también cómo se ha realizado la configuración para obtener el funcionamiento y los resultados deseados.

La familia de micros STM32L consta de una librería comercial facilitada por el fabricante, llamada *Standard Peripherals Library,* completamente gratuita y descargable desde la web de STMicroelectronics, que facilita el uso de los periféricos del microcontrolador. La librería de periféricos presenta la estructura de ficheros que se muestra en la Figura 33.

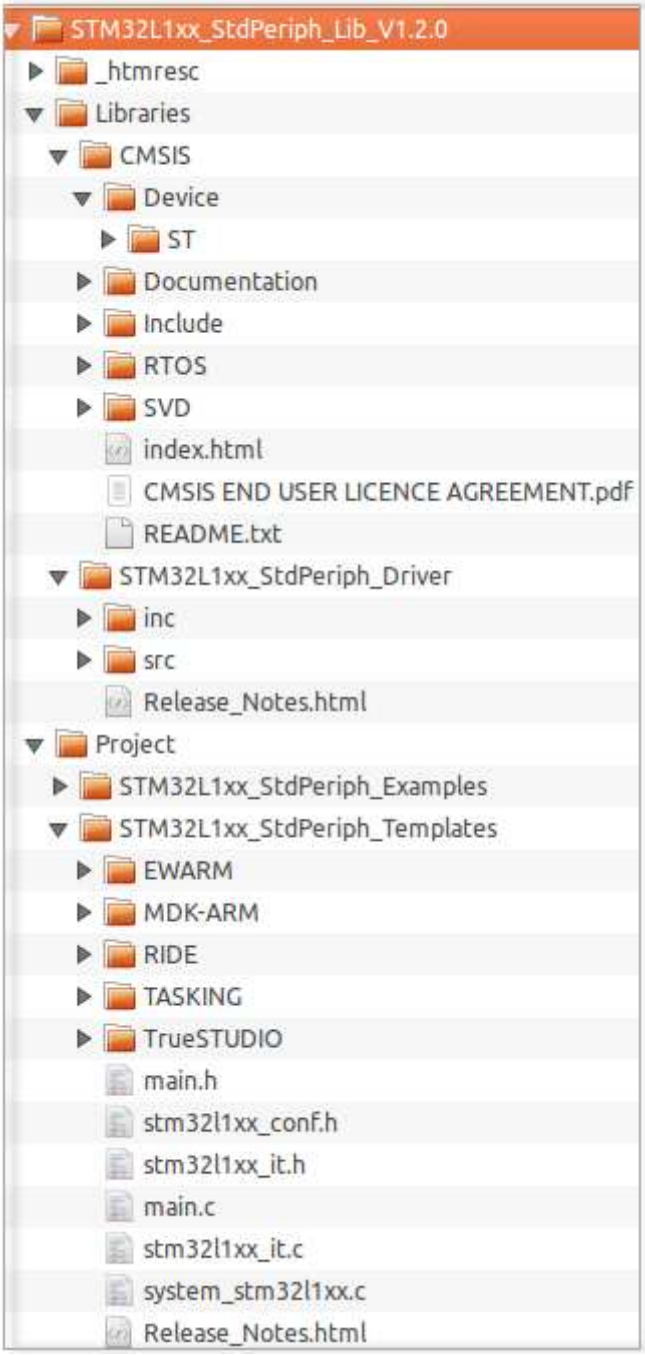

Figura 33 Estructura de ficheros de la librería de periféricos

Esta librería ofrece 3 niveles de abstracción que incluyen:

- Un completo mapa de los registros necesarios con la declaración en lenguaje C de todos los bits, campo de bits y registros. Esto evita tener que realizar una laboriosa tarea de definición y permite ahorrar tiempo sobre todo en las fases iniciales del proyecto.

- Un completo set de rutinas y estructuras de datos que cubren todas las funcionen realizables por los periféricos
- Un set de ejemplos de uso y funcionamiento de los periféricos.

Cada driver consiste en un set de funciones que cubren todas las funcionalidades que puede desempeñar cada periférico [21].

Una de las principales ventajas de la Librería Estándar de Periféricos es que realiza una detección de errores en tiempo de ejecución al llevar a cabo una validación de los valores de entrada de las funciones lo cual aporta robustez al código final.

Es importante destacar que los drivers de la librería están desarrollados con un API común lo que estandariza la estructura de los drivers, las funciones y los nombres de los parámetros. La librería está codificada en "Strict ANSI-C" lo que hace que sea independiente de la herramienta de desarrollo utilizada.

Los ficheros que utiliza la librería estándar de periféricos se detallan a continuación en la Figura 34:

Nombre del Fichero	Descripción
stm32l1xx_conf.h	Fichero de configuración de periféricos. Permite al usuario habilitar o deshabilitar la inclusión de los ficheros de cabecera de cada periférico.
stm32l1xx_ppp.h	Fichero de cabecera del periférico PPP Contiene las definiciones de las variables y funciones referentes al periférico PPP
stm32l1xx_ppp.c	Fichero fuente con el código en lenguaje C del periférico PPP
stm32l1xx_it.h	Fichero de cabecera que contiene los prototipos de los controladores de interrupciones
stm32l1xx_it.c	Fichero de código fuente con el Servicio de Interrupción de Servicio (ISR) para las excepciones del Cortex M3. Permite al usuario añadir ISRs para los periféricos usados.

Figura 34 Ficheros de configuración de la librería de periféricos

4.1.2 Como usar la librería:

Los primeros pasos que se deben seguir tras la creación del proyecto es la configuración de los ficheros CMSIS (Cortex Microcontroller Software Interface Standard). Estos ficheros se encargan de proporcionar una capa de acceso estandarizado a los periféricos:

- stm3211xx.h: Es el fichero de cabecera de CMSIS Cortex M3 STM32Llxxxx de la capa de acceso. Este fichero es el único que hay que incluir en el código de la aplicación, típicamente en el main.c. Permite especificar qué familia es a la que pertenece el micro en uso. En el caso de la placa *STM32l-Discovery*, se cuenta con un micro de la familia "Medium-density", que cuenta con una memoria de programa de entre 64 y 128 Kbyte. Para ello se habilita la línea de código correspondiente a la familia del micro usado dejándose las otras dos comentadas:

  #define STM32L1XX_MD

  También es necesario editar la línea de código que establece si se va a usar la librería para controlar los periféricos o no:

  #define USE_PERIPH_LIBRARY

  Adicionalmente, este fichero permite modificar ciertos parámetros como puede ser la frecuencia del cristal HSE, definir estructuras de datos y mapeo de direcciones para todos los periféricos, declaración de registros de periféricos y definiciones de bit y definición de Macros para acceso del registro de periféricos.

- system.stm3211xx.h: Fichero de cabecera de CMSIS Cortex M3 STM32Llxx de la capa de acceso

- system.stm3211xx.c: Fichero fuente de CMSIS Cortex M3 STM32l.1 xx de la capa de acceso

- startup_stm3211xx_md.s: Fichero Startup para la familia de microcontroladores "STM32L Ultra Low Power Medium density"

- stm32l1xx_conf.h: Fichero de cabecera en el que se selecciona qué periféricos van a ser utilizados por la aplicación. Es necesario comentar las inclusiones de los ficheros de cabecera de los periféricos que no se van a usar.

Una vez seleccionadas las diferentes opciones y configuraciones de los ficheros CMSIS ya se está en posición de usar los drivers de los periféricos definidos en la librería para construir la aplicación. Es importante conocer la nomenclatura utilizada en la librería para facilitar su comprensión. En adelante nos referiremos al periférico de forma genérica como PPP.

El uso de los periféricos a través de la librería consta de cinco pasos principales:

1.- Como paso previo a la configuración de cualquier periférico es necesario habilitar su reloj. Para ello se debe llamar a una de las siguientes funciones dependiendo de a qué bus de datos está conectado el periférico:

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_PPPx, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx, ENABLE);
```

2.- Se debe declarar una estructura del tipo PPP_InitTypeDef. Esta estructura permite inicializar una o más instancias del periférico PPP. Por ejemplo:

```
PPP_InitTypeDef PPP_InitStructure
```

3.- Es necesario definir los valores que se desean configurar en la variable PPP_InitStructure. Estos valores deben ser permitidos acorde con la definición de la estructura. En los ficheros de cabecera (*.h) de cada periférico se pueden consultar los valores permitidos para cada campo de la estructura.

Esto se puede realizar de 2 maneras:

- Se pueden informar todos los miembros de la estructura:

```
PPP_InitStructure.member1 = val1;
PPP_InitStructure.member2 = val2;
PPP_InitStructure.memberN = valN;
```

Estas definiciones se pueden condensar en una sola línea del tipo:

```
PPP_InitTypeDef PPP_InitStructure = {val1, val2,..., valN}
```

- o bien se puede inicializar con los valores por defecto con la función PPP_StructInit(...) y luego modificar específicamente los valores de los miembros necesarios:

```
PPP_StructInit (&PPP_InitStructure);
PPP_InitStructure.memberX = valX;
PPP_InitStructure.memberY = valY;
```

4-. Inicialización del periférico PPP llamando a la función PPP_Init(...). Esta función modifica los registros de configuración del periférico acorde con los valores introducidos en la estructura de inicialización que se le pasa como argumento.

```
PPP_Init(PPP, &PPP_InitStructure);
```

5.- En este punto el periférico está inicializado con los valores seleccionados y solo resta habilitarlo para que entre en funcionamiento con la función PPP_Cmd(...).

```
PPP_Cmd(PPP, ENABLE);
```

Esta función también permite deshabilitar el periférico introduciendo el estado DISABLE como argumento.

Nota: Si se desea devolver al periférico los valores de configuración por defecto se puede usar la función *PPP_DeInit(..)*

```
PPP_DeInit(PPP);
```

4.1.3 Configuración y uso de Periféricos:

En esta sección se describe como se han configurado cada uno de los periféricos del micro utilizados.

4.1.3.1 Relojes (RCC):

En el stm32l152xx hay diferentes fuentes de reloj (internos y externos) así como un PLL (lazo de seguimiento de fase). A continuación se describen las características de cada uno de ellos [16]:

- HSI (*High-Speed Internal*): Reloj interno de 16 Mhz a través de un oscilador RC. Puede ser usado como *System Clock* o como entrada al PLL. No requiere de componentes externos.
- MSI (*Multi-Speed Internal*): Reloj interno a través de un oscilador RC que proporciona siete frecuencias: 65.536 kHz, 131.072 kHz, 262.144 kHz, 524.288 kHz, 1.048 MHz, 2.097 Mhz (valor por defecto) y 4.194 Mhz. El reloj MES es usado como System Clock después del *Reset*, durante el arranque desde los modos Stop, o Standby low power. El valor por defecto con el que arranca es 2.097 MHz.
- LSI (*Low-Speed Internal*). Es un reloj RC de 37 kHz y se utiliza para el *IWDG (Independent Watchdog)* o como reloj para el RTC (Real Time Clock).
- HSE (*High-Speed External*). Proporciona desde 1 hasta 24 MHz de un oscilador de cristal. También funciona como reloj del PLL o el RTC.
- LSE (*Low-Speed External*). Reloj que proporciona 32.768 kHz a partir de un resonador cerámico o un cristal externo. Utilizado como fuente para el reloj RTC y el reloj del LCD.
- PLL. La fuente de reloj proviene del HSI o el HSE. Sirve como System Clock (entre 2 y 24 MHz) o para generar los 48 MHz para el periférico USB.

Tras el reinicio del micro, se utiliza el MSI como reloj de sistema cuya frecuencia se aplica a todos los componentes. Todos los periféricos se encuentras desactivados excepto la SRAM, la memoria Flash y el JTAG. Los GPIOs (General Input/Output pins) se inicializan en modo *input floating*.

Una vez completado el arranque se debe seleccionar una fuente de reloj para el *System Clock*. Dado que no se han especificado criterios previos para la frecuencia de funcionamiento se usará la frecuencia por defecto del MSI (2,097 MHz). Dado que no se va a aplicar ningún pre-escalado a los buses de datos de los periféricos AHB y APB, estos trabajaran también con la frecuencia del MSI.

Como ya se indicó en la sección anterior es necesario inicializar los relojes de los periféricos que se van a utilizar. Para ello se ha creado la función *initRCC()* que realiza las siguientes acciones:

- Activa el reloj del HSI y espera hasta que se encuentra operativo.
- Inicializa los relojes de los periféricos conectados al bus AHB con el uso de la función *RCC_AHBPeriphClockCmd()*.
- Inicializa los relojes de los periféricos conectados a los buses APB1 y APB2 con las funciones *RCC_APB1PeriphClockCmd()* y *RCC_APB2PeriphClockCmd().*

En la figura a continuación se puede observar la distribución de los diferentes periféricos del micro STM32L152RBT6 según los buses a los que están conectados:

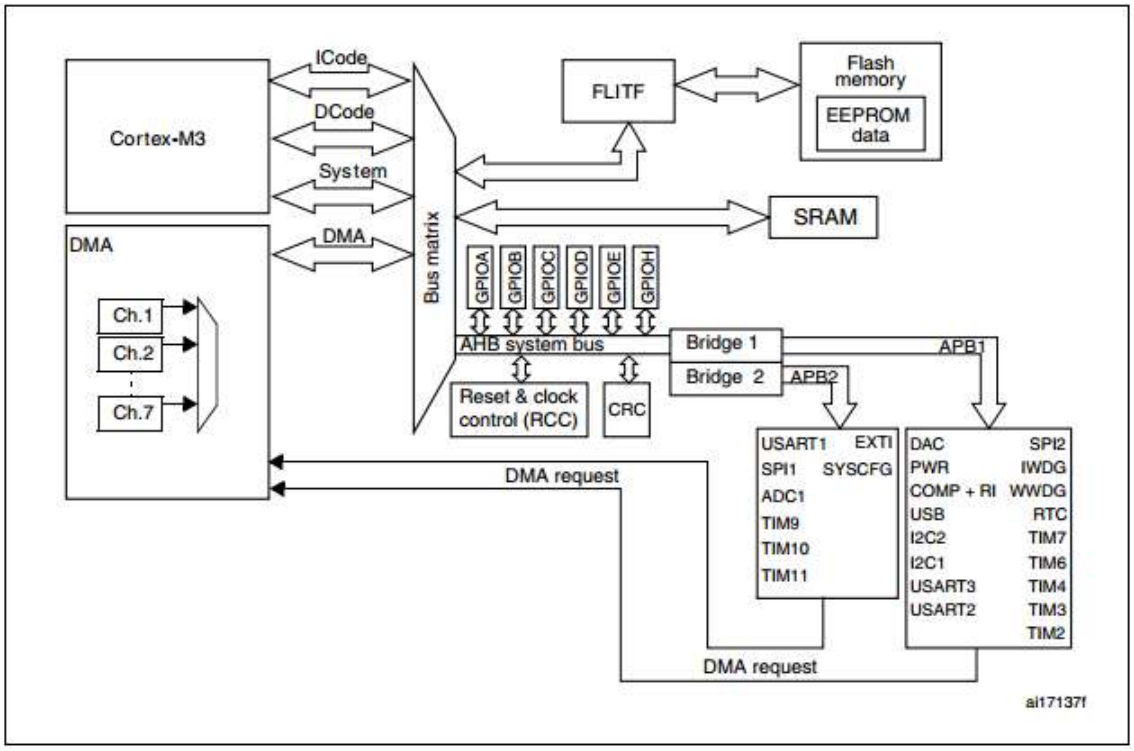

Figura 35 Arquitectura del sistema para micro STM32L152RBT6

4.1.3.2 GPIOs (Pines de Entrada/Salida de Propósito General)

Una vez iniciados los relojes hay que iniciar los pines de la placa para que se comporten como deben ya que son los encargados de controlar las Entradas/Salidas y de gestionar la interacción con ciertos componentes hardware de la placa como los LEDs, el LCD o el circuito de medición de intensidad.

La mayoría de pines de la placa STM32L-Discovery se pueden configurar como entrada o salida y se pueden conectar tanto a los puertos GPIOs como a notros periféricos para desempeñar funciones alternativas. Como norma general los pines se nombran en función al puerto GPIO que se conectan, por ejemplo PA0 corresponde al pin 0 del puerto A.

En la Figura 36 siguiente se muestran las diferentes opciones que se pueden seleccionar para los pines del micro utilizado.

Figura 36 Pinout del micro STM32L152RBT6

Dependiendo del uso que se quiera dar a cada puerto y de las características del hardware que se conecte a ellos, se puede configurar por software y de forma individual con diferentes modos de funcionamiento. Estos modos son:

- Entrada (floating, pull-up o pull-down)
- Salida (open-drain, push-pull)
- Analógico (entrada/salida)
- Alternate function (open-drain, push-pull)

En el desarrollo realizado, los pines de la placa se han configurado de la siguiente manera:

- LCD: Se conectan en modo "Alternative Function":
  - Puerto A: Pin 1, 2, 3, 8, 9, 10, 15
  - Puerto B: Pin 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15
  - Puerto C: Pin 0, 1, 2, 3, 6, 7, 8, 9, 10, 11
- ADC: Se conecta en modo analógico:
  - Puerto A: Pin 4
- LEDs: Se conectan en modo salida Push-Pull:
  - Puerto B: Pin 6, 7
- Botones: Se conectan en modo entrada Push-Pull:
  - Puerto A: Pin 0
- USART1: Se conecta en modo "Alternative Function":

- Puerto A: Pin 9, 10

Para la inicialización de los GPIOs se ha desarrollado la función *init_GPIOs* que se encarga de asignar los valores deseados a la estructura de inicialización de los GPIOs, *GPIO_InitStructure*, y los carga en los registros de configuración con la función *GPIO_Init().*

Como ejemplo, se muestra la inicialización de los pines destinados a los LEDs:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(LD_PORT, &GPIO_InitStructure);
```

Se ha optado por asignar los pines 9 y 10 del Puerto A a la USART impidiendo así que se usen para el LCD ya que un mismo pin no puede desempeñar dos funciones alternativas al mismo tiempo. La solución a la que se ha llegado para poder disponer de ambos periféricos es de conmutar su uso dependiendo del momento, asignándolos a la USART cuando se necesita establecer comunicación con el PC y cambiarlos cuando se quiere mostrar algo por el LCD. Tal y como se ha definido el funcionamiento de la aplicación, no existe ningún momento en el que ambos periféricos entren en funcionamiento simultáneamente.

4.1.3.3 Timers:

Se ha diseñado el uso de 2 de los Timers del micro para cronometrar la ejecución del proceso de cifrado/descifrado.

El timer del micro consiste en un contador de 16 bits con un registro auto-recargable. Los timers se pueden configurar para realizar la cuenta de forma ascendente, descendente o ambas. La configuración de los timers permite asignar

un pre-escalado que divide la frecuencia del reloj por un factor entre 1 y 65536 (16 bits).

Ambos timers se han configurado en cascada. Esta configuración consiste en asignar a uno de los timers el rol de master y al otro el de esclavo. El master comienza a contar con una determinada frecuencia establecida por sus parámetros de configuración. Una vez que alcanza el límite establecido para la cuenta, lanza un evento al otro timer, el esclavo, que incrementa su cuenta en 1. En todo momento la cuenta del timer que actúa como esclavo está condicionada a la del master.

Para esta aplicación se han usado los timer TIM2 y TIM3 del micro como master y esclavo respectivamente.

Para establecer la frecuencia del TIM2 se ha aplicado un pre-escalado de 2 sobre la frecuencia del reloj del sistema, correspondiente al APB2 al que se encuentra conectado el TIM2, obteniendo así una frecuencia entrada de 1MHz Esto quiere decir que el TIM2 incrementará su cuenta cada 1µs. Se ha definido su periodo de recarga en 1000. Esto quiere decir que el evento de recarga ocurrirá cada:

$$T_{TIM2}=\frac{1}{f_{TIM2}}=\frac{1}{f_{clk}/_{TIM2.Period}}=\frac{1}{1MHz/_{1000}}=1ms$$

La estructura de inicialización del periférico se ha configurado de la siguiente manera para conseguir este comportamiento:

```
TIM_TimeBaseStructure.TIM_Period = TIM2_Period;
TIM_TimeBaseStructure.TIM_Prescaler = 2;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
```

Dado que la salida del TIM2 está definida en modo PWM, se ha configurado el parámetro *TIM_Pulse* del TIM2 con el valor 500 para lograr que la señal de salida del TIM2 tenga un ciclo de carga de 50%.

La salida del TIM2 (ITR1) se utiliza como estímulo de entrada del TIM3 que tendrá una frecuencia de entrada de 1KHz.

Se establece la condición de master para el TIM2 y de esclavo para el TIM3 con las siguientes funciones:

```
TIM_SelectMasterSlaveMode(TIM2, TIM_MasterSlaveMode_Enable);
TIM_SelectMasterSlaveMode(TIM3, TIM_MasterSlaveMode_Enable);
TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);
TIM_SelectSlaveMode(TIM3, TIM_SlaveMode_Gated);
```

La medida de Tiempo de ejecución del AES se realiza activando los timers con la función *TIM_Cmd(TIMx, ENABLE)* únicamente durante el algoritmo de encriptación deteniéndolos también con la mismo función (usando *DISABLE* como argumento) en las fases anteriores de la aplicación donde se realizan configuraciones y en las fases posteriores donde se llevan a cabo principalmente las comunicaciones y el procesamiento de las medidas.

Se ha considerado que los tiempos de ejecución de las pruebas no excederán el tiempo máximo de medida de los TIM con esta configuración en cascada. Este tiempo máximo se alcanza cuando la cuenta del TIM3 alcanza el límite. En este caso se ha dejado el valor máximo que corresponde aproximadamente a los 65,5 segundos. En el supuesto de necesitar medir tiempos superiores a este valor sería posible habilitar otro más de los timers del micro como esclavo, a su vez, del TIM3 (esto permitiría medir tiempos de hasta 50 días!).

El valor total de la medida de tiempo de ejecución se obtiene leyendo los valores de las cuentas de los 2 timers. Debido a la configuración de las frecuencias de ambos temporizadores resulta sencillo construir el valor ya que los incrementos del TIM3 corresponden con los ms (milisegundos) mientras que los incrementos del TIM2 corresponden a los µs (microsegundos). El resultado se mostrará como:

$$t_{ejecución}(ms) = Cuenta_{TIM2} + \frac{Cuenta_{TIM3}}{1000}$$

4.1.3.4 Medidas de Consumo

Una de las características más importantes de la placa STM32L-Discovery y probablemente el motivo de mayor peso por el cual se eligió esta placa frente a otras de capacidades similares es el sistema embebido de medición de corriente que trae. Este sistema consta de 2 piezas fundamentales, el circuito de medición de corriente y el ADC aunque también se ha incorporado el *Acceso Directo a Memoria* (DMA) para realizar la consulta de las conversiones del ADC más rápidamente y garantizar el correcto funcionamiento:

- Circuito de medición de corriente:

La esencia del principio de medida del consumo (IDD), consiste en la medida del valor del voltaje en la parte superior del divisor de voltaje, formado por las dos resistencias situadas entre la fuente de alimentación de 3.3 V. y el pin VDD de la MCU. Dependiendo del modo en que se esté funcionando, la aplicación usa R o (2000+2) x R, como el valor de la resistencia equivalente, cerrando o abriendo K1. En modo Run, la corriente está en el rango de los miliamperios, K1 está cerrado y la resistencia equivalente es R. En los modos de bajo consumo, la corriente está en el orden de los amperios, K1 está abierto y la resistencia equivalente es 2002xR. Se puede consultar el esquemático del circuito en la Figura 37.

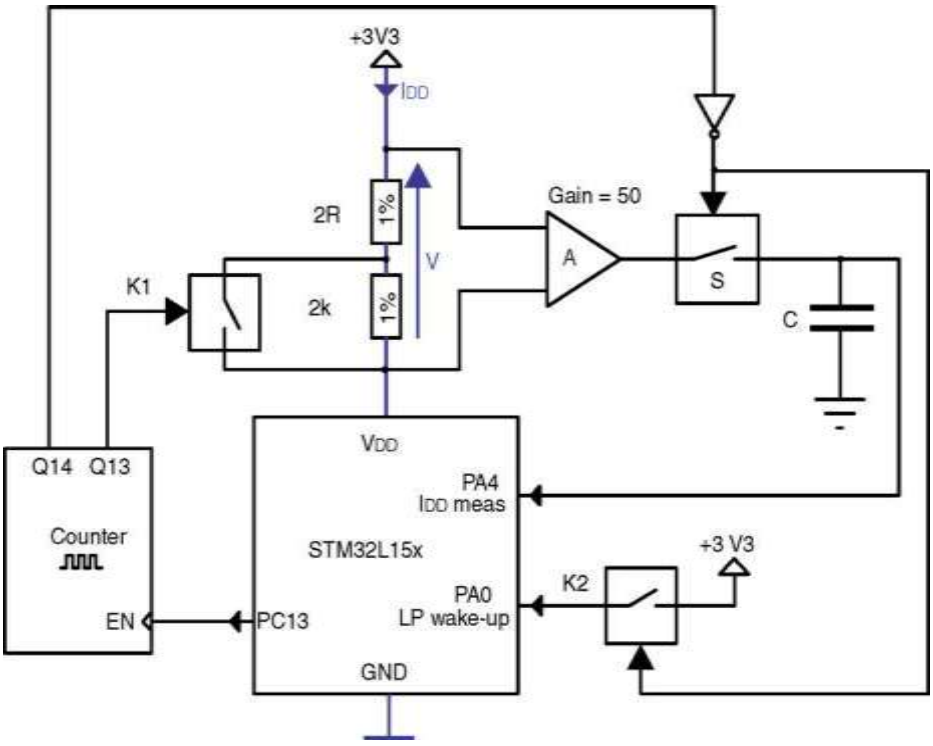

Figura 37 Diagrama del circuito de medición de Icc

La resistencia se conecta entre las entradas diferenciales del amplificador operacional (A), con ganancia fija, que amplifica el voltaje (V) presente en las resistencias. A la salida del amplificador operacional se dispone una etapa de muestreo y retención y se conecta a una entrada analógica de la CPU, como entrada del ADC (conversor Analógico-Digital).

Solamente en los modos de bajo consumo, el pin PC13 de la STM32L15x habilita un contador que dirige el timing de medida, mientras el microcontrolador está funcionando.

Mientras el microcontrolador está en uno de los modos de ahorro de consumo, se carga un condensador (C) con el voltaje amplificado por el operacional. El microcontrolador puede muestrear posteriormente el valor del voltaje del condensador, proporcional al consumo de corriente de la MCU, en modo de bajo consumo. El conmutador S está abierto cuando se arranca el dispositivo para mantener intacta la carga acumulada en el condensador, cuando el micro está en modo bajo consumo.

La precisión en la medida del consumo de corriente se mejora teniendo en cuenta la corriente en vacío del amplificador operacional. Almacenar este valor en la memoria de datos, es importante, para minimizar posteriormente los errores de medida. Es recomendable realizar esta operación cuando se empieza la evaluación para obtener una mejor precisión

- Conversor Analógico–Digital (ADC)

El uso que se le da en la aplicación al ADC consiste en transformar a un valor digital el nivel de tensión que devuelve el circuito de medición de consumo de la placa. Este circuito se conecta al canal 4 del ADC1 del micro (se usa la nomenclatura ADC1 aunque el STM32l152RBT6 solo tiene un ADC). Para ello se ha configurado el ADC en modo Continuo, es decir, toma una medida inmediatamente después de terminar la anterior.

Debido a la alta velocidad de las conversiones ha sido necesario crear un buffer que almacene las medidas de consumo hasta que sean enviadas al PC a través de la UART para su procesamiento. Para este fin se ha usado otro periférico del micro, el DMA (*Direct Memory Access*) que mueve la cuenta del ADC a una variable en memoria.

Para poder interpretar el resultado de la transformación es necesario calcular la relación de transformación:

$$Relación_{Conv} = \frac{3{,}3V}{2^{12}} x1000 = 0{,}806mV$$

El valor final de voltaje consumido por el micro medido en milivoltios (mV) será el resultado de la conversión del ADC multiplicado por esta relación de conversión.

Acceso Directo a Memoria (DMA):

Este periférico permite una transferencia de alta velocidad entre los periféricos y la memoria del micro sin que éste tenga que realizar ninguna acción. Esto permite que los recursos de la CPU queden libres para el resto de operaciones.

El DMA contiene un total de 12 canales, 7 para el DMA1 y otros 5 para DMA2 dedicados al control del acceso a memoria de uno o más periféricos. Son configurables por software y permiten la transmisión de diferentes tamaños de datos (byte (8 bits), half-word (16 bits) y Word (32 bits)) [16].

Para controlar las peticiones de los diferentes canales es capaz de establecer prioridades entre ellos.

Dependiendo de la configuración, la transmisión a través del DMA genera 3 flags que permiten controlar el flujo de datos, una a la mitad de la transmisión (*DMA Half Transfer*), otra al final de la transmisión (*DAM Transfer Completed*) y otra en caso de error (*DMA Transfer Error*).

El DMA permite el acceso, como origen o como destino, a la memoria Flash, a la SRAM y a los buses APBH, APB1 y APB2 con lo que permite no solo la transmisión de datos entre los periféricos y la memoria y viceversa sino que también permite el envío de periférico a periférico y el envío de memoria a memoria.

En el contexto de este proyecto se ha utilizado para transferir las medidas realizadas por el ADC a una variable de memoria que actúa como buffer de envío al PC. La generación de medidas resulta más rápida que la velocidad de envío que tiene la UART con lo que es necesario almacenar los resultados para poder enviarlos de forma completa y que no se pierda ninguno por desbordamiento.

4.1.3.5 USART

El Receptor/Transmisor Asíncrono Universal o UART (en el caso del micro STM·"L utilizado es USART ya que también es capaz de realizar comunicaciones síncronas) es el periférico encargado de establecer la comunicación con otros dispositivos, en este caso el PC.

La comunicación se establece entre la UART del micro y la del ordenador de forma externa a través de un cable. Para poder establecer una comunicación bidireccional es necesario conectar al menos 2 pines de ambas UARTs:

- RX: Pin de Recepción
- TX: Pin de Transmisión

La UART es un elemento que se encarga principalmente de 2 acciones, por un lado convierte la información que le facilita el software en una secuencia de bits que es capaz de transmitir y por otro lado es capaz de recibir estas secuencias de bits y convertirlas en bytes para pasárselas al software.

En esta comunicación se transmiten la información por bytes y cada byte se envía dentro de una secuencia que siempre comienza con una señal baja (0) (start bit) seguida de los 8 bits de información y termina en una señal alta (1). Los bits de información se envían como señales altas (1) o bajas (0). Entre cada secuencia se debe establecer una señal de reposo que corresponde a una señal alta continua. También se pueden realizar comunicaciones con 9 bit de información aunque en este proyecto se ha optado por una comunicación sencilla de 8 bits. Este protocolo de comunicación queda representado en la Figura 38.

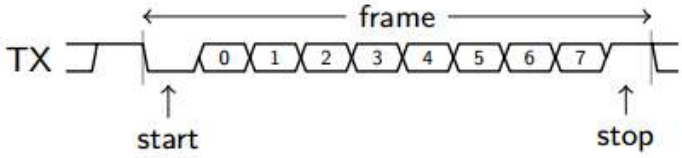

Figura 38 Protocolo de comunicación

.

La transición entre la señal de reposo (alta) y el *start bit* (baja), define el comienzo de cada secuencia de información y supone la única referencia temporal entre ambos dispositivos ya que no existe una señal de reloj común. Cada dispositivo mantiene de forma independiente una señal de reloj acordada entre ambos (o un múltiplo de esta) que normalmente se denomina *relación de Baudios*.

Los micros de la familia STM32L15X cuentan con 5 UARTs aunque para la aplicación desarrollada solo se ha utilizado la USART1 para conectar con el PC.

Al igual que el resto de periféricos descritos en esta memoria, se ha realizado una configuración a través de la *Librería Estándar de Periféricos*. Tras inicializar su reloj (la UART corresponde al bus APB2) es necesario definir la estructura de inicialización con los parámetros de la comunicación que se quiere establecer. Para ello se define dicha estructura de la siguiente manera:

```
USART_InitStructure.USART_BaudRate = BaudRate;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;


USART_Init(USART1, &USART_InitStructure);
```

Adicionalmente es necesario configurar los Pines que van a actuar como TX y RX. Para ello se establecen en modo "función Alternativa" los pines 9 y 10 del puerto A:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1 );
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1 );
```

Con esto queda inicializada la UART1 del micro y solo quedaría activarla con la función *UART1_Cmd.*

La librería comercial provee de todas las funciones necesaria para controlar la comunicación a través de la UART pero se ha optado por crear una librería personaliza donde las funciones de envío y recepción de información se han simplificado. Esta simplificación ha sido posible dada la sencillez en la comunicación a realizar. Esto también ayuda a reducir el tamaño del código, que aunque no se hayan dado problemas de almacenamiento ni de tamaño, siempre es una optimización positiva.

Esta librería personalizada se compone de 6 funciones de las cuales tres se destinan a la inicialización y configuración de la UART y las otras 3 para la comunicación:

- UART1_Init: Habilita el reloj y configura les pines TX y RX
- UART1_Config: Asigna los valores a los parámetros de comunicación (baudios, tamaño de palabra, bits de parada, paridad…)
- UART1_Cmd: Habilita o deshabilita la UART
- usart_rec: Recupera un byte del Buffer de lectura
- usart_snd: Escribe un byte en el buffer de salida
- usart_snd_str Escribe una secuencia de bytes en el buffer de salida.

El fichero de cabecera incluye las definiciones de estas funciones así como las inicializaciones de las variables y los valores de las constantes necesarias para configurar la comunicación:

4.1.3.6 *DISPLAY* LCD

El display LCD de la placa se ha utilizado principalmente para mostrar las selecciones que se realizan y las medidas que se toman de consumo. Otro uso que se debe destacar es el de elemento de depuración ya que resulta de mucha ayuda mostrar por el display información durante el desarrollo del código.

Para su configuración se cuenta con un driver específico para el LCD de la placa STM32L-Discovery que permite de forma sencilla inicializarlo y usarlo con un set de funciones muy completo.

El LCD requiere de la configuración de numerosos pines de los puertos de propósito general (GPIOs) configurados en "función alternativa". En la Figura 39 se muestra la distribución de los pines para el LCD.

STM32L152	LCD					
Name	Pin	COM3	COM2	COM1	COM0	Name
PA1	1	1N	1P	1D	1E	LCDSEG0
PA2	2	1DP	1COLON	1C	1M	LCDSEG1
PA3	3	2N	2P	2D	2E	LCDSEG2
PB3	4	2DP	2COLON	2C	2M	LCDSEG3
PB4	5	3N	3P	3D	3E	LCDSEG4
PB5	6	3DP	3COLON	3C	3M	LCDSEG5
PB10	7	4N	4P	4D	4E	LCDSEG6
PB11	8	4DP	4COLON	4C	4M	LCDSEG7
PB12	9	5N	5P	5D	5E	LCDSEG8
PB13	10	BAR2	BAR3	5C	5M	LCDSEG9
PB14	11	6N	6P	6D	6E	LCDSEG10
PB15	12	BAR0	BAR1	6C	6M	LCDSEG11
PB9	13	COM3				LCDCOM3
PA10	14		COM2			LCDCOM2
PA9	15			COM1		LCDCOM1
PA8	16				COM0	LCDCOM0
PA15	17	6J	6K	6A	6B	LCDSEG12
PB8	18	6H	6Q	6F	6G	LCDSEG13
PC0	19	5J	5K	5A	5B	LCDSEG14
PC1	20	5H	5Q	5F	5G	LCDSEG15
PC2	21	4J	4K	4A	4B	LCDSEG16
PC3	22	4H	4Q	4F	4G	LCDSEG17
PC6	23	3J	3K	3A	3B	LCDSEG18
PC7	24	3H	3Q	3F	3G	LCDSEG19
PC8	25	2J	2K	2A	2B	LCDSEG20
PC9	26	2H	2Q	2F	2G	LCDSEG21
PC10	27	1J	1K	1A	1B	LCDSEG22
PC11	28	1H	1Q	1F	1G	LCDSEG23

Figura 39 Pinout del Display LCD en placa STM32L-Discovery [21]

Para establecer los pines en función alternativa, debemos habilitar el reloj de los puertos GPIOs e inicializar los pines necesarios con dicha configuración. Esto se realiza con la función *Init_GPIOs_LCD().*

Para poder comenzar a mostrar información por el LCD y al igual que todos los demás periféricos, se debe habilitar el reloj del LCD e inicializarlo con la estructura de inicialización correspondiente. El LCD tiene al LSE como fuente de reloj a 32KHz. A continuación se muestra el código de inicialización del LCD:

```
LCD_InitStruct.LCD_Prescaler = LCD_Prescaler_1;
LCD_InitStruct.LCD_Divider = LCD_Divider_31;//
LCD_InitStruct.LCD_Duty = LCD_Duty_1_4;
LCD_InitStruct.LCD_Bias = LCD_Bias_1_3;
LCD_InitStruct.LCD_VoltageSource = LCD_VoltageSource_Internal;
```

Existe una función en el driver que realiza estas dos acciones así como la configuración de parámetros como el contraste, tiempos de inactividad, etc, para que el periférico quede totalmente operativo. Esta función se llama: *LCD_GLASS_Init.*

Por último el driver nos ofrece una serie de funciones para mostrar información por el display:

- *LCD_GLASS_DisplayString.* Escribe un *string (cadena de caracteres)* en el *display* LCD. Recibe como parámetro un puntero a un *string*.
- *LCD_GLASS_Clear()*. Esta función borra la memoria RAM del LCD.
- *LCD_GLASS_ScrollSentence().* Se utiliza esta función cuando se desea mostrar un *string* en modo *scroll*.
- *LCD_GLASS_DisplayStrDeci().* Esta función escribe un *char* en la memoria RAM del LCD. Requiere que se le pase una matriz de caracteres en ASCII.

El LCD también tiene la posibilidad de mostrar la información parpadeando gracias a la función: *LCD_BlinkConfig().*

4.1.3.7 NVIC (Nested Vectored Interrupt Controller)

Se trata del periférico encargado de controlar y administrar las interrupciones y su prioridad. Es capaz de gestionar hasta 45 canales de interrupción asignando los niveles de prioridad (hasta 255 niveles).

La arquitectura ARM cuenta con controladores o *handlers* específicos para las interrupciones así como otros que son específicos del procesador. Estos controladores se encuentran definidos en una tabla donde se asignan las posiciones de memoria a cada uno.

Existen numerosos controladores definidos en la arquitectura:

```
Reset_Handler                    DMA1_Channel6_IRQHandler
NMI_Handler                      DMA1_Channel7_IRQHandler
HardFault_Handler                ADC1_IRQHandler
MemManage_Handler                EXTI9_5_IRQHandler
BusFault_Handler                 TIM1_BRK_TIM15_IRQHandler
UsageFault_Handler               TIM1_UP_TIM16_IRQHandler
SVC_Handler                      TIM1_TRG_COM_TIM17_IRQHandler
DebugMon_Handler                 TIM1_CC_IRQHandler
PendSV_Handler                   TIM2_IRQHandler
SysTick_Handler                  TIM3_IRQHandler
WWDG_IRQHandler                  TIM4_IRQHandler
PVD_IRQHandler                   I2C1_EV_IRQHandler
TAMPER_IRQHandler                I2C1_ER_IRQHandler
RTC_IRQHandler                   I2C2_EV_IRQHandler
FLASH_IRQHandler                 I2C2_ER_IRQHandler
RCC_IRQHandler                   SPI1_IRQHandler
EXTI0_IRQHandler                 SPI2_IRQHandler
EXTI1_IRQHandler                 USART1_IRQHandler
EXTI2_IRQHandler                 USART2_IRQHandler
EXTI3_IRQHandler                 USART3_IRQHandler
EXTI4_IRQHandler                 EXTI15_10_IRQHandler
DMA1_Channel1_IRQHandler         RTCAlarm_IRQHandler
DMA1_Channel2_IRQHandler         CEC_IRQHandler
DMA1_Channel3_IRQHandler         TIM6_DAC_IRQHandler
DMA1_Channel4_IRQHandler         TIM7_IRQHandler
DMA1_Channel5_IRQHandler
```

Figura 40 Controladores de interrupciones de STM32L15X

En este proyecto únicamente se ha hecho uso de los handlers de la USART1 y del Systick.

USART1_IRQn:

La configuración del NVIC define la prioridad que se le asigna a las interrupciones generadas y también de activar y desactivar dichas interrupción. Esto se realiza a través de la configuración de la estructura de inicialización, por ejemplo para USART1:

```
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Cuando se produce la interrupción, la ejecución principal se suspende y se ejecuta la RAI (Rutina de Atención a la interrupción) Una vez ejecutada se devuelve el control a la aplicación principal.

La RAI que se ha desarrollado para la USART1 salta cada vez que se detecta una información en el buffer de salida o de entrada y proceder al envío o recepción de dicha información.

Las RAI se han definido en el fichero *stm32l1xx_it.c.* Para el caso de la USART1 es:

```
void USARTx_IRQHANDLER(void)
{
        if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
        {
        RxBuffer[RxCounter++] = (USART_ReceiveData(USART1) & 0x7F);
        if(RxCounter == NbrOfDataToRead)
                {
                USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
                }
        }
```

```
if(USART_GetITStatus(USART1, USART_IT_TXE) != RESET)
        {
        USART_SendData(USART1, TxBuffer[TxCounter++]);
        if(TxCounter == NbrOfDataToTransfer)
                {
                USART_ITConfig(USART1, USART_IT_TXE, DISABLE);
                TxCounter=0;
                }
        }
}
```

La otra interrupción que se ha usado en el proyecto es la del Systick.

Los micros con arquitectura Cortex-M3 poseen un timer dedicado para generar ticks. El uso que se le ha dado en este proyecto a este timer es la generación de retrasos (delays).

La configuración es muy sencilla ya que cuenta con una función específica de configuración para asignar la frecuencia a las que saltarán las interrupciones:

```
SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000);
```

En este caso la variable *RCC_Clocks.HCLK_Frequency* corresponde a los ciclos de reloj del sistema de un segundo, con lo que la configuración mostrada corresponde a una interrupción cada milisegundo (ms). Esto quiere decir que cada milisegundo se ejecutará la RAI asociada al *SysTick_Handler* que se encarga de reducir el valor de la variable *TimingDelay* que contiene el número de milisegundos que dura el *delay*. Una vez que la variable llegue a cero la aplicación continuará.

4.1.4 Implementación del algoritmo AES

La implementación del algoritmo AES, tanto el cifrado como el descifrado se han realizado de 2 maneras, una usando un tamaño de palabra de 8 bits y otro con un tamaño de 32 bits. Para facilitar la comprensión del código y su manejabilidad, se

ha creado una librería específica, *AES.c*, que aglutina todas las funciones específicas del algoritmo AES.

Usando esta librería solo es necesario invocar cada función en el orden correcto.

La librería cuenta con un fichero de cabecera que contiene todas las definiciones de las funciones: AES.h.

A continuación se muestra el seudocódigo del cifrado y del descifrado donde se invoca a las funciones de la librería:

- Cifrado 32 bits:

```
addRoundKey32(0);
for(ronda=1;ronda<numeroRondas;ronda){
        subBytes32();
        shiftRows32();
        mixColumns32();
        addRoundKey32(ronda);}
subBytes32();
shiftRows32();
addRoundKey32(numeroRondas);
```

- Descifrado 32 bits

```
addRoundKey32(numeroRondas);
for(ronda=numeroRondas - 1;ronda>0;ronda--){
        invShiftRows32();
        invSubBytes32();
        addRoundKey32(ronda);
        invMixColumns32();}
invShiftRows32();
invSubBytes32();
addRoundKey32(0);
```

- Cifrado 8 bits

```
addRoundKey8(0);
for(ronda=1;ronda<numeroRondas;ronda++){
        subBytes8();
        shiftRows8();
        mixColumns8();
        addRoundKey8(ronda);}
subBytes8();
shiftRows8();
addRoundKey8(numeroRondas);
```

- Descifrado 8 bits

```
addRoundKey8(numeroRondas);
for(ronda=numeroRondas - 1;ronda>0;ronda--){
        invShiftRows8();
        invSubBytes8();
        addRoundKey8(ronda);
        invMixColumns8();}
invShiftRows8();
invSubBytes8();
addRoundKey8(0);
```

4.2 Desarrollo en Matlab:

A continuación se describe e funcionamiento del script desarrollado en *MATLAB* que hace de interfaz con el usuario además de encargarse de las siguientes funcionalidades:

- Lectura y procesado de los datos de entrada: Abre el fichero de datos que se van a procesar (cifrado o descifrado) leyendo en cada iteración solo el número de elementos necesario para completar el tamaño de bloque
- Permite elegir al usuario las diferentes opciones de funcionamiento del algoritmo. Se debe elegir si el proceso será de cifrado o de descifrado y si se

utilizará un tamaño de palabra de 8 bits o de 32 bits. Estas opciones se muestran por pantalla y se seleccionan por teclado.

- Permite realizar la conexión con el micro para intercambiar información. Envía al micro las opciones de funcionamiento y recibe la información de las medidas de consumo realizadas.
- Procesa la información recibida y la almacena para su posterior tratamiento.
- Muestra gráficamente el resultado de las medidas de consumo enviadas por el micro.

En la Figura 42 se muestra el flujograma del desarrollo en Matlab donde se pueden ver los diferentes pasos por los que avanza el proceso.

El script comienza pidiendo que el usuario seleccione los diferentes modos de funcionamiento que permite la aplicación

La primera selección corresponde a qué elemento se va a utilizar para visualizar las medidas como se muestra en la Figura 41.

Figura 41 Matlab - Selección de entorno de trabajo

La opción 1 lanza un proceso de cifrado que funciona de forma *standalone* en la placa. Este proceso realiza la encriptación de un bloque y una clave almacenados en memoria y muestra las medidas de consumo directamente en el LCD de la placa. Tras terminar la encriptación del texto plano en memoria, muestra el mensaje "SUCCESS" por el LCD. Este sistema simplemente pretende mostrar las medidas por pantalla y demostrar que el proceso de toma de medidas de consumo funciona correctamente.

La otra opción, la 2, es la de usar la herramienta Matlab como interfaz. Una vez seleccionada esta opción se permite elegir el modo de funcionamiento de AES entre el cifrado y el descifrado implementados en 32 o en 8 bits ()

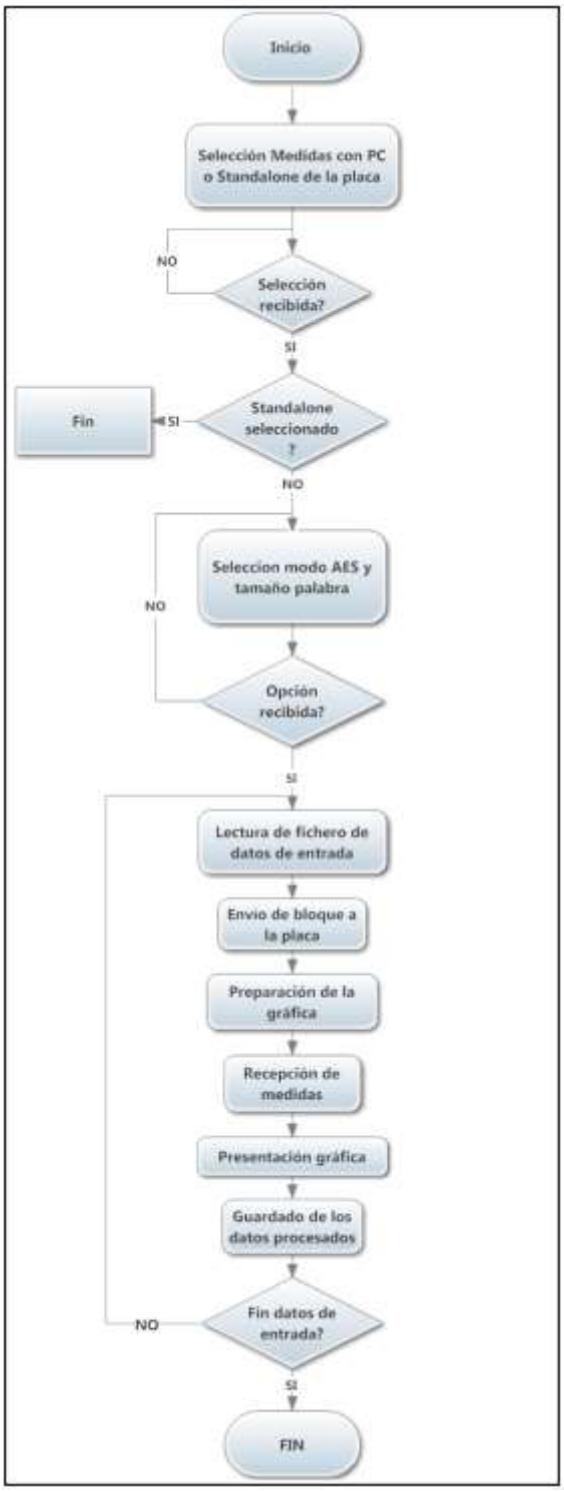

Figura 42 Flujograma desarrollo en Matlab

La selección del modo de funcionamiento se hace con el menú de opciones que se muestra en la Figura 43.

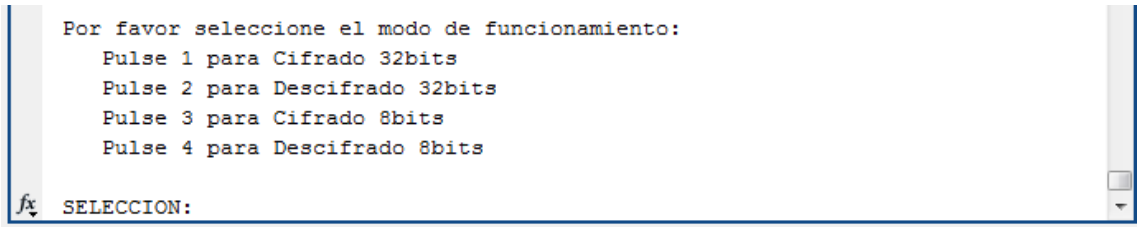

Figura 43 Selección modo AES

Una vez selecciona el modo, el script permite introducir por teclado la clave que se va a usar en el cifrado o en el descifrado ()

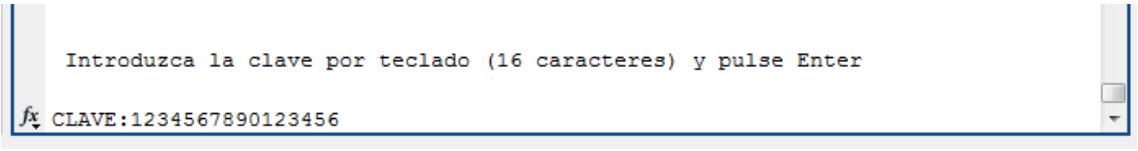

Figura 44 Obtención de la clave

Una vez introducida la clave el script accede al fichero de texto que se desea procesar y lee el primer bloque de 16 caracteres y lo envía por puerto serie a la placa. Tras hacer esto, se queda en espera de que la placa le envíe el texto procesado, los tiempos de ejecución y las medidas de consumo.

Estos tres elementos los almacena en tres ficheros de datos separados para poder acceder a esta información posteriormente.

El script, a medida que recibe los datos de consumo, crea una gráfica donde muestra la evolución del consumo a lo largo de la ejecución del AES. Las gráficas y los datos obtenidos serán descritos en mayor detalle en el siguiente capítulo.

Cuando se han terminado de recibir todos los datos de las medidas de consumo, el script comienza de nuevo leyendo los siguientes 16 bytes del fichero de texto a procesar. En caso de que el siguiente bloque tenga menos de 16 elementos porque se haya alcanzado el fin del fichero, el proceso completará el bloque con ceros (*padding*) dado que el AES debe trabajar con tamaño de bloques fijos.

Tal y como se muestra en la Figura 45, una vez que se alcanza el fin del fichero de datos y se completa la última vuelta del proceso, se muestra un mensaje por pantalla indicando el fin de la ejecución.

Figura 45 Fin del proceso de encriptación

4.3 Ejecución y Medidas

La ejecución del algoritmo de encriptación y la toma de medidas en la placa se puede dividir en 4 partes principales por orden de ejecución:

1. Inicialización y configuración de los componentes de la placa.
2. Comunicación con el PC para seleccionar los modos de funcionamiento y adquirir la clave y el bloque de texto plano
3. Ejecución del algoritmo AES obteniendo medidas de consumo durante todo el proceso de encriptación y medidas de tiempo tanto para cada función como para el total del algoritmo.
4. Envío de información al PC

Se han realizado ejecuciones para los 4 modos seleccionables dependiendo de si se trata de cifrar o descifrar y si se hace en 8 bits o en 32 bits.

Cada ejecución ha consistido en el procesamiento de 10 bloques con 16 bytes cada uno. Tanto las medidas de consumo como las de tiempo se han tomado solo durante la ejecución del algoritmo de encriptación que es el foco de nuestro análisis, deteniendo los dispositivos de medida durante las demás fases.

En la Figura 46, se puede observar la gráfica que se genera en Matlab a partir de las medidas realizadas en la placa. En se pueden observar las variaciones en el consumo del micro durante la ejecución. Los valores de consumo obtenidos tienen un valor entorno a los 20,5 mV lo que se corresponde a una corriente de consumo de 205µA. este valor resulta un poco más bajo del esperado ya que en la documentación del micro se especifica una corriente de carga en modo Run de 250µA.

No se aprecia una diferencia entre la ejecución procesada con 8 bits y la ejecutada con 32 bits en cuanto al nivel de tensión aunque como es lógico, el hecho de que

la ejecución sea más corta en el tiempo hace que, con mismos niveles de tensión, se consuma menos potencia.

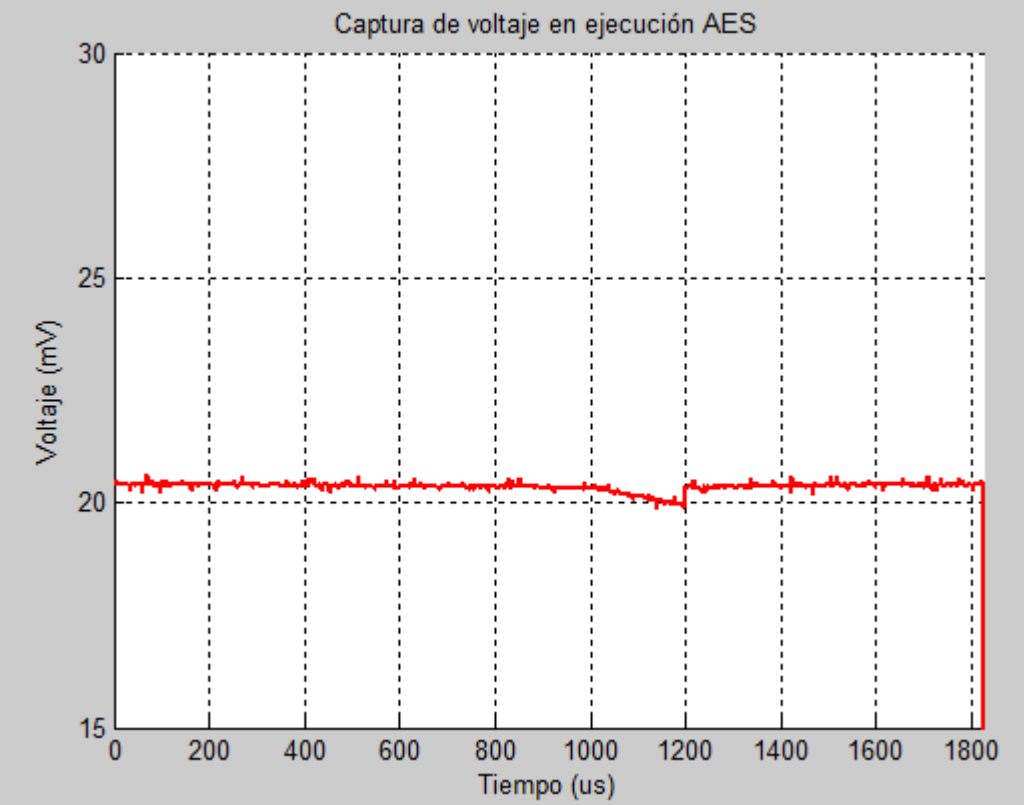

Figura 46 Grafica de consumo para cifrado AES32 bits

Se puede observar en las figuras generadas por Matlab o consultando directamente los ficheros de datos donde se almacenas las medidas que el grado de resolución del sistema de medida es suficiente para apreciar gradientes de tensión muy reducidos. En modo *Run*, en el cual se abre el divisor de tensión del circuito de medida como ya se explicó en *sección 4.1.3.4*. Idealmente estaríamos hablado de detectar, con esta configuración, variaciones de 2.9 µA.

A continuación, en la Figura 47, se muestra una ampliación de la anterior figura donde se puede observar el grado de detalle que se alcanza.

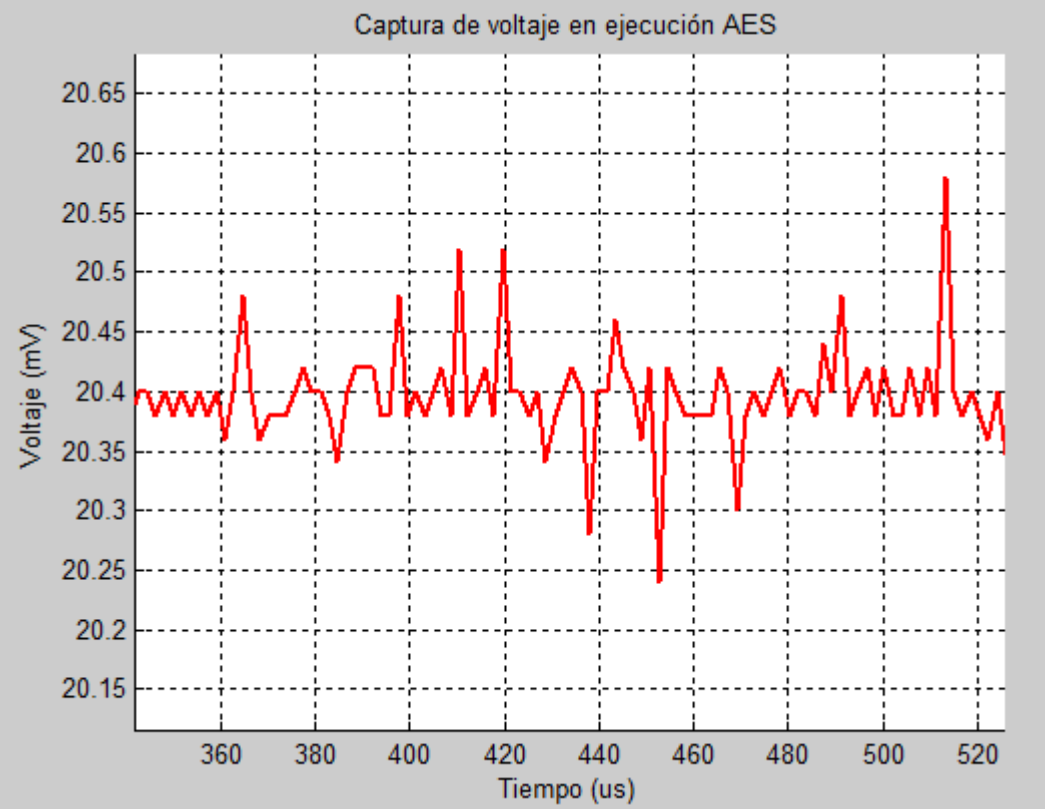

Figura 47 Ampliación trazas de consumo

Si la corriente fuese menor a estos valores, sería suficiente con configurar el circuito de medida de corriente para que la resistencia equivalente sea de 2002Ω en vez de lo 2Ω del modo *Run (activar comparador desde el micro).*

Las medidas de tiempo se han realizado con el objetivo principal de analizar si las implementaciones en 32 bits mejoran la velocidad de la ejecución.

Función	Implementación 8 Bits Tiempo ejecución por ronda (µs)	Implementación 32 Bits Tiempo ejecución por ronda (µs)
KeyExpansion	1229,70	529,1034
AddRoundKey	66,53	24,459
SubBytes	58,40	33,927
ShiftRows	10,91	8,7579
MixColumns	80,60	55,23
InvShiftRow	12,09	9,2313

InvSubBytes	57,23	34,1637
InvMixColumns	1104,00	473,4

Figura 48 Tiempos de ejecución 8bits vs. 32bits

Como se puede apreciar en la Figura 48 los tiempos han mejorado notablemente al hacer uso de toda la capacidad del micro y procesar el algoritmo con un tamaño de palabra de 32 bits.

Tanto las medidas de tiempo y de consumo, como los bloques de texto cifrado, se almacenan en cada ejecución en la carpeta del entorno de trabajo de Matlab seleccionada por el usuario, dentro de los ficheros indicados en la Figura 49:

Contenido	Nombre del fichero
Tiempos	AES_time.dat
Voltaje	AES_Vconsumo.dat
Texto Procesado	AES_salida.txt

Figura 49 Ficheros de resultados

5. Conclusiones

En esta sección se muestran las conclusiones a las que se han llegado tras la realización de este proyecto.

El objetivo principal se definió, no solo como la realización de un sistema de medidas sino también, el análisis de la capacidad de los componentes de la placa STM32L-Discovery para realizar dichas medidas. La aplicación permite al usuario, interactuando únicamente con el PC, enviar a la placa tanto la clave como el texto a procesar y obtener de vuelta el texto cifrado, las medidas de consumo durante la aplicación y un desglose de los tiempos empleados por las diferentes funciones que componen el AES.

El desarrollo presentado en esta memoria, demuestra que la placa STM32L Discovery y sus componentes son capaces que manejar las mediciones de consumo de este tipo de sistemas criptográficos, información que resulta de gran valor dentro del estudio de los ataques por canal lateral, donde se enmarca este proyecto.

Dentro de los objetivos secundarios o de apoyo al proyecto, se encontraba el estudio del algoritmo criptográfico AES, lo cual se ha llevado a cabo, adquiriendo no solo el conocimiento suficiente para implementarlo en el micro, sino también para poner en práctica mejoras sobre el mismo, como la ejecución en 32 bits.

Uno de las barreras que ha sido necesaria salvar, ha sido la de aprendizaje del micro y la arquitectura ARM ya que lo estudiado en la carrera anteriormente consistía en la familia 8051 desarrollando las aplicaciones en ensamblador. Tras concluir el proyecto se ha logrado programar el micro en lenguaje C para conseguir la aplicación deseada. La familiarización con el micro ha sido necesaria principalmente ya que para lograr el correcto funcionamiento de la aplicación ha sido necesario hacer uso de múltiples elementos del microcontrolador.

Con esto, se puede concluir que los objetivos de este proyecto se han llevado a cabo satisfactoriamente.

6. Presupuesto

En esta sección se calcularán los costes que han supuesto la realización de este proyecto. Se calcularán los costes por personal, por el uso de dispositivos Hardware y las licencias y compras de herramientas software. Tras el cálculo particular de cada categoría se procederá a obtener el presupuesto total del proyecto.

6.1 Coste del Personal

Las actividades desempeñadas por el personal encargado de este proyecto se pueden desglosar en las siguientes:

- Estudio del algoritmo AES (2 semanas – 60 horas)
- Estudio del micro y su arquitectura (2 semanas – 60 horas)
- Implementación del algoritmo AES (3 semanas – 90 horas)
- Implementación de sistema de medidas y comunicación con PC (2 semanas – 60 horas)
- Realización de pruebas (1semana – 30 horas)
- Elaboración de la memoria (4 semanas – 120 horas)
- Tutelaje del proyecto (40 horas)

6.2 Coste de *Hardware*

Los elementos hardware que se han utilizado para este proyecto son:

- Ordenador personal sobremesa: 600€
- Placa STM32L-Discovery: 15€
- Cable TTL-232R 3V3WE: 17€

6.3 Costes de *Software*

Las aplicaciones software que se han utilizado para el proyecto son:

- Atollic true Studio Lite con licencia gratuita.
- Microsoft Office 2013: versión Hogar y Estudiante: 112€.
- MatLab: Licencia gratuita de la Universidad Carlos III.

6.4 Presupuesto total

Además de los gastos de las 3 categorías anteriores se debe añadir un 20% en gastos indirectos.

El presupuesto total del proyecto se detalla en la plantilla mostrada en la Figura 50.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1. - Autor: Brandán Currás Paz

2.- Departamento: Departamento de Tecnología Electrónica

3. - Descripción del Proyecto

- Título	Sistema de medidas de consumo para aplicaciones criptográficas en microcontrolador ARM Cortex-M3
-Duración (meses)	6
Tasa de costes indirectos	20%

4. - Presupuesto total del Proyecto (€):

23.080,52

5. -Desglose presupuestario (costes directos)

Personal

Apellidos y nombre	Categoría	Dedicación (hombres mes)	Coste hombre mes	Coste (Euros)
Currás Paz, Brandán	Analista	2,057	5250	10799,25
Currás Paz, Brandán	Programador	1,371	2625	5399,31
Currás Paz, Brandán	Ingeniero pruebas	0,229	3937	1200,00
Mengibar Pozo, Luis	Ingeniero Senior	0,305	5250	1600,00
			TOTAL:	18998,56

EQUIPOS

Descripción	Coste Euro	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Placa STM32L-Discovery	15	100	6	60	1,5
Ordenador Personal	700	100	6	60	70
Cable TTL-232R 3V3WE	17	100	6	60	1,7
				TOTAL:	**73,2**

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

Subcontratación de tareas

Descripción	Empresa	Coste
	TOTAL:	0,00

OTROS COSTES DIRECTOS DEL PROYECTO

Descripción	Empresa	Coste imputable
Licencia Windows 7		0,00
Licencia Microsoft Office 2013		112,00
Material oficina		50,00
	TOTAL:	162,00

6. -Resumen de costes

Categoría	Presupuesto costes totales
Personal	18998,56
Equipos	73,20
Subcontratación de tareas	0,00
Otros costes directos	162,00
Costes indirectos	3846,75
TOTAL:	23080,52

Figura 50 Presupuesto total del proyecto

7. Trabajos Futuros

Se puede considerar este proyecto como el primer paso dentro del estudio de ataques de canal lateral a sistemas criptográficos ya que presenta al usuario con la herramienta para realizar las medidas. Como pasos posteriores se debe realizar un estudio de las medidas a nivel cualitativo para lograr extraer conclusiones sobre el sistema investigado.

Adicionalmente se proponentes los siguientes puntos como evolución del desarrollo realizado:

1. Se propone la posibilidad de añadir a este sistema de medida la capacidad de adquirir lecturas de temperatura del micro ya que cuenta con un sensor de temperatura integrado.
2. Se propone también, la implementación de los diferentes modos de funcionamiento de AES ya que este proyecto se ha centrado únicamente en el ECB (*Electronic Code Book*) sin entrar a analizar ninguno de los otros tres.
3. Se propone analizar el impacto en el consumo en ejecuciones del algoritmo criptográfico configurando el micro en modo *Low Power Run*.

8. Anexos

8.1 Anexo I: Índice de Figuras

8.2 Anexo II: Referencias

[1] S. F. Fernández, «La criptografía clásica,» *Sigma,* pp. 119-142, 2004.

[2] U. Granada, «Universidad de granada,» [En línea]. Available: http://www.ugr.es/~anillos/textos/pdf/2010/EXPO-1.Criptografia/02a22.htm. [Último acceso: 23 Septiembre 2013].

[3] «Wikipedia - Enigma,» Agosto 2013. [En línea]. Available: http://es.wikipedia.org/wiki/Enigma_%28m%C3%A1quina%29. [Último acceso: Noviembre 2013].

[4] «Wikipedia - Criptología,» 25 Noviembre 2013. [En línea]. Available: http://es.wikipedia.org/wiki/Criptolog%C3%ADa. [Último acceso: 30 Noviembre 2013].

[5] G. ÁLVAREZ, «Criptografía: si no existiera, habría que inventarla,» Instituto de Física Aplicada (CSIC), Diciembre 2011. [En línea]. Available: http://www.fgcsic.es/lychnos/es_ES/articulos/criptografia_si_no_existiera_habria_que_inventarla. [Último acceso: Diciembre 2013].

[6] D. Coppersmith, «The Data Encriptyon Standard (DES) and its strengthagainst attacks,» IBM, 1994.

[7] «Announcing the Advance Encryption Standard,» (NIST), National Institute of Standards and Technology, 2001.

[8] r. e. referencia, *Historia del AES (Rondas).*

[9] Revisar, *velocidades de ejecución en algoritmos finalistas.*

[10] NIST, «AES Algorithm (Rijndael) Information,» Febrero 2001. [En línea]. Available: http://csrc.nist.gov/archive/aes/index.html. [Último acceso: Septiembre 2013].

[11] E. Zabala, «Rijndael Inspector,» 2008. [En línea]. Available: http://www.formaestudio.com/rijndaelinspector/. [Último acceso: mayo 2013].

[12] «Wikipedia - Advanced Encryption Standard,» Agosto 2013. [En línea]. Available: http://es.wikipedia.org/wiki/Advanced_Encryption_Standard. [Último acceso: Septiembre 2013].

[13] A. M. Muñoz, Seguridad Europea para EEUU - Algoritmo Rijndael, 2004.

[14] «Wikipedia - Modos de operación de una unidad de cifrado por bloques,» 12 marzo 2013. [En línea]. Available: http://es.wikipedia.org/wiki/Modos_de_operaci%C3%B3n_de_una_unidad_de_cifrado_por_bloques. [Último acceso: agosto 2013].

[15] STMElectronics, «018775 - STM32L-DISCOVERY,» 2011.

[16] STMElectronics, «RM0038 - Reference Manual,» 2012.

[17] ARM, «ARM - Corte-M3,» 2013. [En línea]. Available: http://www.arm.com/products/processors/cortex-m/cortex-m3.php. [Último acceso: Noviembre 2013].

[18] STMicroelectronics, «AN3371 - Using the hardware real-time clock (RTC),» 2012.

[19] «Using Direct Memory Access (DMA) in STM32 projects,» November 2012. [En línea]. Available: http://www.embedds.com/using-direct-memory-access-dma-in-stm23-projects/. [Último acceso: Noviembre 2013].

[20] FTDI Chip, «TTL to USB Serial Converter Range of Cables Datasheet,» 2010.

[21] STMElectronics, «STM32L1xx_StdPeriph_Driver V1.1.1 (release notes),» 2012.

[22] STMElectronics, «TA0340 - STM32L Cortex-M3 microcontroller for usage in low-power healthcare applications,» STMicroelectronics , 2011.

[23] G. Brown, Discovering the STM32 Microcontroller, 2012.